

CART[®] 6.0

User's Guide

Dan Steinberg and Mikhail Golovnya
Salford Systems

4740 Murphy Canyon Rd. Suite 200
San Diego, California 92123, USA

619.543.8880 TEL

619.543.8888 FAX

www.salford-systems.com

Developers of TreeNet[®], MARS[®] RandomForests[®] and other
award-winning data mining and predictive analytics tools

© Salford Systems, 2002-2007

Copyright

Copyright 2002-2007, Salford Systems; all rights reserved worldwide. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise without the express written permission of Salford Systems.

Limited Warranty

Salford Systems warrants for a period of ninety (90) days from the date of delivery that, under normal use, and without unauthorized modification, the program substantially conforms to the accompanying specifications and any Salford Systems authorized advertising material; that, under normal use, the magnetic media upon which this program is recorded will not be defective; and that the user documentation is substantially complete and contains the information Salford Systems deems necessary to use the program.

If, during the ninety (90) day period, a demonstrable defect in the program's magnetic media or documentation should appear, you may return the software to Salford Systems for repair or replacement, at Salford Systems option. If Salford Systems cannot repair the defect or replace the software with functionally equivalent software within sixty (60) days of Salford Systems receipt of the defective software, then you shall be entitled to a full refund of the license fee. Salford Systems cannot and does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free. Salford Systems disclaims any and all liability for special, incidental, or consequential damages, including loss of profit, arising out of or with respect to the use, operation, or support of this program, even if Salford Systems has been apprised of the possibility of such damages.

Citations

The proper citations for CART technology and this software are:

Breiman, Leo, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification and Regression Trees*. Pacific Grove: Wadsworth, 1984.

Steinberg, Dan and Phillip Colla. *CART—Classification and Regression Trees*. San Diego, CA: Salford Systems, 1997.

Steinberg, Dan and Mikhail Golovnya. *CART 6.0 User's Manual*. San Diego, CA: Salford Systems, 2006.

Trademarks

CART is a registered trademark of California Statistical Software, Inc. and is exclusively licensed to Salford Systems. StatTransfer is a trademark of Circle Systems. DBMS-Copy is a trademark of Conceptual Software. All other trademarks mentioned herein are the property of their respective owners.

Table of Contents

Copyright.....	1
Limited Warranty.....	1
Citations.....	2
Trademarks.....	2
INTRODUCING CART 6.0	9
Introduction	10
What's New in CART 6.0?	14
About this Manual.....	20
INSTALLING AND STARTING CART	23
Installing and Starting CART 6.0	24
Minimum System Requirements	24
Recommended System Configuration.....	24
Installation Procedure From CD-ROM	25
Ensuring Proper Permissions	26
Starting and Running CART	26
Licensing CART	26
Preparing Your Data for CART	28
Setting up Working Directories	28
READING DATA	31
General Comments	32
Accessing Data from Salford Systems Tools	32
Variable Naming.....	35
Reading Excel Files	36
CART BASICS	39
CART Tutorial.....	40
CART Desktop.....	41
About CART Menus	41
About CART Toolbar Icons.....	42
Opening a File	43
Setting Up the Model	45
Tree Navigator.....	48
Viewing the Main Tree	56
Viewing Sub-trees.....	58
Assigning Labels and Color Codes	59
Printing the Main Tree	60
Tree Summary Reports	61
Gains Chart/Cumulative Accuracy Profile	61
Terminal Nodes	63
Variable Importance	64

Misclassification	66
Prediction Success or Confusion Matrix	67
Detailed Node Reports	68
Terminal Node Report	73
Saving the Navigator/Grove File	74
More Navigator Controls	74
CART Text Output	75
Displaying and Exporting Tree Rules	76
Scoring Data	77
New Analysis	78
Saving the Command Log	78
CLASSIFICATION TREES	81
Building Classification Trees	82
The Model tab	85
The Categorical tab	92
The Testing Tab	95
The Select Cases tab	100
The Best Tree tab	102
The Method Tab	104
The Advanced Tab	111
The Cost Tab	116
The Priors tab	118
The Penalty tab	121
Setting Reporting, Random Number and Directory Options	125
Working with Navigators	134
Viewing Auxiliary Variables Information	134
Comparing Children	139
Comparing Learn and Test	139
Saving Navigator Files	140
Printing Trees	141
Overlaying and Printing Gains Charts	143
REGRESSION TREES	145
Building Regression Trees	146
Specifying a Regression Model	146
Tree Navigator	149
Regression Tree Summary Reports	151
Detailed Node Reports	154
Terminal Node Report	158
ENSEMBLE MODELS AND COMMITTEES OF EXPERTS	161
Building an Ensemble of Trees	162
Bootstrap Aggregation and ARCing	162
The Combine Tab	164

SCORING AND TRANSLATING	169
Scoring and Translating Models	170
Navigator Files versus Grove Files	170
Converting a Tree File to a Grove File	172
Scoring CART models	172
Score Data Dialog	173
Output Data Set	175
Score GUI Output for Classification Trees	176
Case Output for Regression Trees	178
Scoring in Command Line	180
Translating CART models	180
Translating in Command Line	182
Exporting and Printing Tree Rules	183
TRAIN-TEST CONSISTENCY (TTC)	185
Optimal Models and Tree Stability	186
HOT SPOT DETECTION	193
Searching for Hot Spots	194
CART BATTERIES	199
Batteries of Runs	200
CART SEGMENTATION	227
Modeling the multi-class target	228
CART Desktop	228
About CART Menus	229
Opening a File	230
Setting Up the Model	232
Tree Navigator	235
Viewing Variable Splits	237
Viewing the Main Splitters	238
Viewing the Main Tree	239
Viewing Sub-trees	242
Assigning Labels and Color Codes	242
Printing the Main Tree	243
Tree Summary Reports	244
Gains Chart	245
Root Splits	247
Terminal Nodes	247
Variable Importance	248
Misclassification	249
Prediction Success	250
Detailed Node Reports	251
Terminal Node Report	256
Saving the Grove File	257

CART Text Output	257
Displaying and Exporting Tree Rules	258
Scoring Data	259
New Analysis	261
Saving Command Log	261
FEATURES AND OPTIONS.....	263
Features and Options	264
Unsupervised Learning and Cluster Analysis.....	264
The Force Split tab	267
The Constraints tab.....	275
Saving and Printing Text Output	283
Memory Management.....	285
Report Writer.....	288
Data Viewer	290
Data Information	291
WORKING WITH COMMAND LANGUAGE	295
Introduction to the Command Language	296
Alternative Control Modes in CART for Windows.....	297
Command-Line Mode.....	298
Creating and Submitting Batch Files	298
Command Log	299
View—Open Command Log	299
File—New Notepad	300
File—Submit Window.....	300
File—Submit Command File.....	301
Command Syntax Conventions	301
Example: A sample classification run.....	302
Example: A sample regression run	305
UNIX/Console Usage Notes.....	310
COMMAND LINE MENU EQUIVALENTS	315
ERRORS AND WARNINGS	319
COMMAND REFERENCE.....	327
ADJUST	328
AUXILIARY	329
BATTERY.....	330
BOPTIONS.....	334
BUILD.....	338
CATEGORY	339
CDF	340
CHARSET	341
CLASS	342

COMBINE	344
DATA	345
DATAINFO	346
DESCRIPTIVE	347
DISCRETE	348
DISALLOW	350
ERROR	352
EXCLUDE	353
FORCE	354
FPATH	355
FORMAT	356
GROUP	357
GROVE	358
HARVEST	359
HELP	361
HISTOGRAM	362
IDVAR	363
KEEP	364
LABEL	365
LCLIST	366
LIMIT	369
LINEAR	371
LOPTIONS	372
MEMO	373
MEMORY	374
METHOD	375
MISCLASS	376
MODEL	377
MOPTIONS	378
NAMES	380
NEW	381
NOTE	382
OPTIONS	383
OUTPUT	384
PARTITION	385
PRIORS	388
PRINT	389
QUIT	390
REM	391
RUN	392
SCORE	393
SAVE	395
SEED	396
SELECT	397
STRATA	398
SUBMIT	399

TRANSLATE.....	400
USE	402
WEIGHT	403
XYPLOT	404
BASIC PROGRAMMING LANGUAGE.....	405
BASIC Programming Language.....	406
Getting Started with BASIC Programming Language	406
BASIC: Overview of BASIC Components	407
LET	407
IF...THEN.....	407
ELSE	407
FOR...NEXT	408
DIM	408
DELETE	409
Operators	409
BASIC Special Variables.....	409
BASIC Mathematical Functions	410
BASIC Probability Functions	411
Missing Values	413
More Examples	413
Filtering the Data Set or Splitting the Data Set	414
DATA Blocks.....	415
Advanced Programming Features.....	415
BASIC Programming Language Commands	416
DELETE Statement.....	416
DIM Statement	417
ELSE Statement.....	418
FOR...NEXT Statement.....	419
GOTO Statement	420
IF. . . THEN Statement	421
LET Statement	422
STOP Statement	423

Intro

Introducing CART 6.0

This chapter provides a brief introduction to CART and this manual, and an overview of new features.

Introduction

Welcome to CART 6.0 for Windows, a robust decision-tree tool for data mining, predictive modeling, and data preprocessing. CART automatically searches for important patterns and relationships, uncovering hidden structure even in highly complex data. CART trees can be used to generate accurate and reliable predictive models for a broad range of applications from bioinformatics to risk management and new applications are being reported daily. The most common applications include churn prediction, credit scoring, drug discovery, fraud detection, manufacturing quality control, and wildlife research. Several hundred detailed applications studies are available from our website at <http://www.salford-systems.com>.

CART uses an intuitive, Windows-based interface, making it accessible to both technical and non-technical users. Underlying the "easy" interface, however, is a mature theoretical foundation that distinguishes CART from other methodologies and other decision trees.

Salford Systems' CART is the only decision-tree system based on the original CART code developed by world-renowned Stanford University and University of California at Berkeley statisticians Breiman, Friedman, Olshen and Stone. The core CART code has always remained proprietary and less than 20% of its functionality was described in the original CART monograph. Only Salford Systems has access to this code, which now includes enhancements co-developed by Salford Systems and CART's originators.

There is only one true CART and Salford Systems in collaboration with CART's creators is the only source for this remarkable technology.

Based on decades of machine learning and statistical research, CART provides reliable performance and accurate results. Its market-proven methodology is characterized by:

A complete system of reliable data analysis

When the CART monograph was first published it revolutionized the emerging field of decision trees. An entire methodology was introduced for the first time that included multiple tree-growing methods, tree pruning, methods to deal with unbalanced target classes, adapting to the cost of learning and the cost of mistakes, self-testing strategies, and cross validation. For the scientifically minded, rigorous mathematical proofs were provided to show that the underlying algorithms were mathematically sound and could be relied upon to yield trustworthy results.

The CART monograph, published in 1984, is now justly regarded as a landmark work and one of the most important mathematical events of the last 30 years. It is one of the most-frequently cited works in machine learning and data mining.

An effective tree-growing methodology

CART introduced several new methods for growing trees, including the Gini and the innovative Twoing method, among others. These methods have proven effective in uncovering productive trees and generating insights into data. To cover a broad variety of problems, CART also includes special provisions for handling ordered categorical data and the growing of probability trees. Important extensions to these core CART methods found in CART 6.0 are discussed below.

A powerful binary-split search approach

CART trees deliberately restrict themselves to two-way splits of the data, intentionally avoiding the multi-way splits common in other methods. These binary decision trees divide the data into small segments at a slower rate than multi-way splits and thus detect more structure before too few data are left for analysis. Decision trees that use multi-way splits fragment the data rapidly, making it difficult to detect patterns that are visible only across broader ranges of data values.

An effective pruning strategy

CART's developers determined definitively that no stopping rule could be relied on to discover the optimal tree. They introduced the notion of over-growing trees and then pruning back; this idea, fundamental to CART, ensures that important structure is not overlooked by stopping too soon. Other decision-tree techniques use problematic stopping rules that can miss important patterns.

Automatic self-test procedures

When searching for patterns in databases it is essential to avoid the trap of "over fitting," that is, of finding patterns that apply only to the training data. CART's embedded test disciplines ensure that the patterns found will hold up when applied to new data. Further, the testing and selection of the optimal tree are an integral part of the CART algorithm. In other decision-tree techniques, testing is conducted only optionally and after the fact and tree selection is based entirely on training data computations.

CART accommodates many different types of real-world modeling problems by providing a unique combination of automated solutions.

Cross Validation and Repeated Cross Validation

Cross validation, one of CART's self-testing methods, allows modelers to work with relatively small data sets or to maximize sample sizes for training. We mention it here because implementing cross validation for trees is extraordinarily challenging and easy to get wrong technically. With CART you get cross validation as implemented by the people who invented the technology and introduced the concept into machine learning. In CART 6.0 we allow you to rerun many replications of cross validation using different random number seeds automatically so that you can review the stability of results across the replications and extract summaries from an averaging of the results.

Surrogate splitters intelligently handle missing values

CART handles missing values in the database by substituting "surrogate splitters," back-up rules that closely mimic the action of primary splitting rules. The surrogate splitter contains information that typically is similar to what would be found in the primary splitter. You can think of the surrogate splitter as an imputation that is customized to the node in the tree in which it is needed and that makes use of other relevant information in the data.

Other trees treat all records with missing values as if the records all had the same unknown value; with that approach all such "missings" are assigned to the same bin. In CART, each record is processed using data specific to that record, allowing records with different data patterns to be handled differently and resulting in a better characterization of the data.

CART 6 also automatically analyzes whether missingness is in itself predictive and will optionally incorporate such findings into the optimal model.

Adjustable misclassification penalties help avoid the most costly errors

CART includes "cost-sensitive" learning so that models developed by CART can incorporate the seriousness of any mistake. In a binary classification problem we often label the outcomes 0 and 1 and, by default, assume that all classification

errors are equally costly. But what if misclassifying a 1 as a 0 (a false negative) is far worse than misclassifying a 0 as a 1 (a false positive)?

CART users can specify a higher “cost” for the more serious mistakes, causing the software to steer the tree away from that type of error. That is, in response to the cost information CART will actually grow a different tree. The greater the cost of a specific kind of mistake the more CART will adjust the tree to avoid the high cost mistakes. Further, when CART cannot guarantee a correct classification, it will try to ensure that the errors it does make are less costly. If credit risks were classified as low, moderate, or high, for example, it would be more costly to misclassify a high-risk borrower as low-risk than as moderate-risk. Traditional data mining tools and many decision trees cannot distinguish between these types of misclassification errors in their model construction processes.

Alternative splitting criteria make progress when other criteria fail

CART includes seven single-variable splitting criteria, Gini, Symgini, Twoing, Ordered Twoing, Entropy and Class Probability for classification trees, and Least Squares and Least Absolute Deviation for regression trees. In addition, we offer one multi-variable or oblique splitting criterion, the Linear Combinations or LC method. CART 6 includes some important extensions to the classic LC method.

The default Gini method frequently performs best, but by no means is Gini the only method to consider in your analysis. In some circumstances the Twoing method will generate more intuitive trees. To help you find the best method CART will optionally test all its methods automatically and summarize the results in tables and charts.

What's New in CART 6.0?

Our goal in developing CART 6.0 has been to help the data analyst be more productive and to make the whole process of developing high performance models faster, easier and more intuitive. We have introduced new ways to shape and control models, new ways to assess the quality of your models, and added tools to report, deploy, and export models for production purposes. This section provides a *brief* and selective overview of the newest features. Complete details are provided in the main body of the manual. For a list of the features introduced in CART 5.0 please see the relevant appendix.

CART-Pro and CART-ProEX

To accommodate a diverse set of user requirements we are now offering three main versions of CART 6.0: the SE or “standard edition,” the Pro or “professional,” and the Pro EX or “professional extended edition” intended for our most demanding users. Features available only in the Pro and ProEX versions are marked throughout the documentation using the following indicators.



CART 6.0 Pro indicator.



CART ProEX indicator.

Groves and Navigators

CART 6.0 now uses only the grove file (.grv) to store model information and no longer creates navigator (.nav or .nv3) files. CART 6.0 will still read your old navigator files so you can continue to view and extract reports from them. You will not need navigator files in the future because CART 6.0 stores all model information in the grove.

Data Preparation and Management

All Salford tools have traditionally offered a comprehensive built-in BASIC programming language for on-the-fly data manipulation. The language includes full flow control in FOR..NEXT loops, GOTOs and array processing. Core capabilities include filtering and deleting records on the basis of simple or complex criteria. New variables can be constructed with the help of more than 50 mathematical and statistical functions and a complete set of logical, text, and arithmetic operators. These functions have been available to assist modelers in adjusting data and focusing on specific data subsets.

Starting in 2006 we have made it easier to use our data processing machinery for the sole purpose of data preparation. You can now read in data in any one of our supported data formats, process the data as required, and then save the results in another data format, without having to conduct any modeling. In other words, you can now use our software as a dedicated data preparation tool.

Descriptive Statistics

Our complete set of statistics, including standard summary statistics, quantiles, and detailed tabulations, continue to be available in a single easy-to-access display. We now also offer an abbreviated version in the traditional *one row per predictor* format.

Also new in CART 6.0 are sub-group statistics based on any segmentation or stratification variable.

Tree Control (e.g., Forced Splits, Constraints



CART 6.0 allows you to dictate the splitter to be used in the root, or in either of the two children of the root. This control is frequently desired by users wanting to impose some modest structure on a tree. You can also specify the split values for both continuous and categorical splitters if you prefer to do so.

A much more sophisticated set of controls is available in CART-ProEX. These controls allow you to pre-specify sets of variables to be used in specific regions of the tree and to determine the order in which splitters appear in the tree. Look for a discussion of the “structured tree” to learn more about this patent-pending feature.

Missing Value Controls and Analysis

CART has always offered sophisticated high performance missing value handling. In CART 6.0 we introduce a new set of missing value analysis tools for automatic exploration of the optimal handling of your incomplete data. On request, CART 6.0 will automatically add missing value indicator variables (MVI), for every variable containing any missing values, to your list of predictors and conduct a variety of analyses using them. For a variable named X1, the MVI will be named X1_MIS and coded as 1 for every row with a missing value for X1 and 0 otherwise. If you activate this control, the MVIs will be created automatically (as temporary variables) and will be used in the CART tree if they have sufficient predictive power.

For categorical variables an MVI can be accommodated in two ways: by adding a separate MVI variable or by treating missing as a valid “level.” Modelers can now experiment to see which works best.

MVIs allow formal testing of the core predictive value of knowing that a field is missing. One of the models CART 6.0 will generate for you automatically is a model using only missing value indicators as predictors. In some circumstances such a simple model can be very accurate and it is important to be aware of this predictive power. Other analyses explore the benefits of imposing penalties on variables that are frequently missing.

Modeling Automation Batteries



Most modelers conduct a variety of experiments, trying different model control parameters in an effort to find the best settings. This is done for any method that has a number of control settings that can materially affect performance outcomes. In our training courses we have regularly recommended conducting such experiments via our scripting language and have shown students how to set up such experiments for the most important controls. In CART 6.0 we have made the process easier yet by packaging our recommended “batteries of models” into batches that the modeler can request with a mouse click. CART-Pro includes a core set of batteries, including batteries for ATOM, MINCHILD, MVI (Missing Value Indicators), and tree-growing methods (RULES). Cross validation can now be repeated with different random number seeds (CVR) and the results can be averaged over a set of CV experiments. See the relevant section in the manual for a complete list of batteries offered.

CART-Pro EX includes a larger set of batteries, including new methods for refining the list of predictors (KEEP list) and assuring greater model stability. These batteries can run hundreds or even thousands of models to help you find a model of suitable performance and complexity (or simplicity).

Modeling Refinement from the Variable Importance List



Once a model is built you can easily refine it by managing the variable importance list. Just highlight the variables you want to keep for the next model and click the “Build New Model” button.

CART-EX provides a higher degree of automation for predictor list refinement (feature extraction) and offers an automated pre-modeling predictor discovery stage. This can be very effective when you are faced with a large number of candidate predictors. In our extensive experiments we have established that automatic predictor discovery frequently improves CART model performance on independent holdout (validation) data.

New Linear Combination Controls

In classic CART, linear combination splits are searched for over all numeric predictors. If an LC splitter is found it is expressed in a form like:

```
If 2.2345 * X1 - .01938 * X2 + .98548 * X3 <= 1.986
then a case goes left
```

Such splitters are difficult to interpret and tend to be used only when interpretability can be sacrificed in favor of accuracy. While a few academic studies have embraced LCs (also known as *oblique* splits), they have largely not been used in practical modeling settings.

Our new controls may not persuade you to make use of LCs but they can help to make trees more interpretable and are likely to also give better results. In CART 6.0 you may specify lists of variables (LC lists) from which any LC can be constructed. Every variable in an LC must appear on a single LC list. Thus, in a credit risk model you might list credit report variables on one list, core demographics on another list, and current income-related variables on a third list. Such LC lists force combinations of variables used in an LC splitter to all be of a specific type. Time series analysts might create a separate LC list for a variable and all its lagged values.

If LC lists contain no more than two predictors then any LCs used in the tree will be of the simplest possible form: a weighted average of two predictors.



CART ProEX includes a new control that allows an LC list to be limited to a specific node size regardless of how many variables are on an LC list. Additionally, we have added controls to limit the number variables allowed in a LC, an improvement adjustment for DOF, as well as an improvement penalty control.



Hot Spot Detection

When the goal of an analysis is to identify especially interesting subsets of the data we may place little value on the overall performance of a model. So long as a model is effective in identifying a high concentration of the class of interest it may not matter to us whether the model exhibits good overall accuracy. We call the process of uncovering especially good segments hot spot detection and the process is fully automated in CART-EX.

Additional Summary Reports:*ROC curves (train/test)*

ROC curves have become a preferred way of summarizing the performance of a model and these are now available for all CART models and ensembles. An estimate of the area under the ROC curve is also produced when cross validation is used to assess model performance.

Learn/Test/Pooled Results

Results can be viewed for either the learn (training) data, the test data, or the aggregate created by pooling the learn and test samples.

Gains Chart: Show Perfect Model

In a gains curve the performance of a perfect model depends on the balance between the "response" and "nonresponse" sample sizes. The "perfect model" reference line helps to put the observed gains curve into proper perspective.

Activity Window

The activity window offers a quick way to access summary statistics, summary graphs, the model setup dialog, a view of the data, and scoring.

User-Controlled Cross-Validation Bins

If you prefer to create your own partition of the data for the purpose of cross validation, you can specify that CART is to use a variable you have created for this purpose. This is most useful when there are repeated observations on a behavioral unit such as person, or a firm, and it is important to keep all records pertaining to such a unit together (either all records are in the training sample or all are in the test sample). User-constructed CV bins are also useful in the analysis of time series or geographically-correlated data.

Repeated Cross-Validation Bins (e.g., BATTERY CVR)

CART produces its cross-validation bins via a randomized partition of the data into the requested number of partitions (or folds). To explore how results might differ as the random partitioning differs, you can request repeated CART runs in which the CV bins are constructed using different random starting points.

Additional Fraction for Auto Validation

Traditionally CART trees are grown on learn (or training) data and evaluated on test data. Because the test data are used to help select the optimal-sized tree, some practitioners prefer to conduct a further model check by evaluating a performance on a never looked at ("holdout") portion of the data. We refer to these holdout data as the validation data.

Improved Probability Trees

In CART 5 probability tree performance was summarized using a version of the Gini splitting criterion. In CART 6 we use the same relative error metric that is used for all other CART splitting rules.

Additional Model Evaluation Methods:

Monte Carlo testing (BATTERY MCT)

Randomization tests can provide useful sanity checks on model performance. With the MCT battery CART takes the dependent variable and randomly shuffles it, exchanging the correct value of the target with the value from another randomly-selected row in the data. Such shuffling should make it very difficult for CART to generate predictive trees. The extent to which trees are still predictive is a measure of the potential over-optimism in the measurement of any tree on the actual data.

Profit display using defined auxiliary variables



"Profit" variables are any variables the modeler is interested in tracking in the terminal nodes. The "profit" tab on the summary window includes tabular and graphical displays of these variables, showing absolute and average node results, and cumulative results based on the ordering of the nodes as determined by the original target variable.



Unsupervised Learning

We believe that Leo Breiman invented this trick but we are not entirely sure. We start with the original data and then make a copy. The copy has each of its columns randomly shuffled to destroy its original correlation structure. CART is then used to try to recognize whether a record belongs to the original data or to the shuffled copy. The stronger the correlation structure in the original data the better CART will do and the terminal nodes may identify interesting data segments.

New Model Translation Formats

In CART 6, we have added Java and PMML to our existing group of model translation languages.

The Predictive Modeling Markup Language (PMML) is a form of XML specifically designed to express the predictive formulas or mechanisms of a data mining model. In CART 6 we conform to PMML release 3.0.



Train-Test Consistency

Classic CART trees are evaluated on the basis of overall tree performance. However, many users of CART are more interested in the performance of specific nodes and the degree to which terminal nodes exhibit strongly consistent results across the train and test samples. The TTC report provides new graphical and tabular reports to summarize train-test agreement.

About this Manual

This User's Guide provides a hands-on tutorial as well as step-by-step instructions to orient you to the graphical user interface and to familiarize you with the features and options found in CART. We have also incorporated command line syntax for our non-GUI Linux and UNIX users.

This manual is not intended to instruct the user on the underlying methodology, but rather to provide exposure to the basics of the CART software application. If you are new to CART and decision trees we think you will find CART an ideal way to learn. After you have become familiar with the nuts and bolts of running CART we recommend that you devote some time to further reading.

The primary source of information about the software's methodology is the main reference manual, *CART—Classification and Regression Trees*, which contains a comprehensive discussion of the conceptual basis and features of CART. As you work through this manual you may find it helpful to consult the main manual for more detailed discussion of some technical terms and concepts.

Additional detailed information about the CART algorithm and the thinking of the authors can be found in the original CART monograph:

Breiman, Leo, Jerome Friedman, Richard Olshen, and Charles Stone.
Classification and Regression Trees. Pacific Grove: Wadsworth, 1984.

The remainder of the Windows User's Guide is organized as follows:

- ◆ Chapter 1: INSTALLING AND STARTING CART
- ◆ Chapter 2: READING DATA
- ◆ Chapter 3: CART BASICS
- ◆ Chapter 4: CLASSIFICATION TREES
- ◆ Chapter 5: REGRESSION TREES
- ◆ Chapter 6: ENSEMBLE MODELS AND COMMITTEES OF EXPERTS
- ◆ Chapter 7: SCORING AND TRANSLATING
- ◆ Chapter 8: TRAIN-TEST CONSISTENCY (TTC)
- ◆ Chapter 9: HOT SPOT DETECTION
- ◆ Chapter 10: CART BATTERIES
- ◆ Chapter 11: CART SEGMENTATION
- ◆ Chapter 12: FEATURES AND OPTIONS
- ◆ Chapter 13: WORKING WITH COMMAND LANGUAGE
- ◆ Appendix I: COMMAND LINE MENU EQUIVALENTS
- ◆ Appendix II: ERRORS AND WARNINGS
- ◆ Appendix III: COMMAND REFERENCE
- ◆ Appendix IV: BASIC PROGRAMMING LANGUAGE



This chapter provides a brief instruction on how to install and start CART, and how to prepare to read the data.

Installing and Starting CART 6.0

This chapter provides instructions for installing and starting CART 6.0 for Windows 2000, Windows 2003, and Windows XP. Although CART 6.0 may run on older versions of the Windows operating system we strongly recommend that you rely on later versions of Windows.

Minimum System Requirements

To install and run CART, the minimum hardware you need includes:

- Pentium processor or similar
- 512 MB of random-access memory (RAM)
This value depends on the "size" of CART you have licensed (32 MB, 64MB, 128MB, 256MB, 512MB, 1GIG, 2GIG). While some versions of CART will run with a minimum of 128MB of RAM, we highly recommend that you follow the recommended memory configuration that applies to the particular version of CART you have licensed. Using less than the recommended memory configuration results in excessive hard drive paging, reducing performance significantly and risking that you will run out of resources quickly, leading to a shut down of the software.
- Hard disk with 40 MB of free space for program files, data file access utility, and sample data files
- Additional hard disk space for scratch files (with the required space contingent on the size of the input data set)
- CD-ROM or DVD drive for installation from external media. Installers may be downloaded from our web and ftp sites, eliminating the need for the CD-ROM drive.
- Windows 2000 /2003 /XP

Recommended System Configuration

Because CART is extremely CPU intensive, the faster your CPU, the faster CART will run. For optimal performance, we strongly recommend that CART run on a machine with a system configuration equal to, or greater than, the following:

- Pentium 4 processor running 1.0+ GHz.
- Amount of RAM needed depends on the "size" of CART you have licensed (128MB, 256MB, 512MB, 1GIG, 2GIG). While several versions of CART will run with a minimum of 128MB of RAM, we urge you to follow the recommended memory configuration that applies to your version of CART.

Using less than the recommended memory configuration results in hard drive paging, reducing performance significantly.

- Hard disk with 40 MB of free space for program files, data file access utility, and sample data files
- Additional hard disk space for scratch files (with the required space contingent on the size of the input data set)
- CD-ROM or DVD drive to install from external media. All CART installation files including documentation are also available over internet connections.
- Windows 2000 /2003 /XP
- 2 GIG of additional hard disk space available for virtual memory and temporary files

Installation Procedure From CD-ROM

To install CART:

1. Insert the CD labeled CART 6.0 into your CD-ROM drive. If Autorun is enabled on your system, the installation starts automatically and you can skip steps 2 and 3.
2. From the start menu, select Run.
3. In the Run dialog box, type D:\SETUP (substituting the appropriate drive letter of your CD-ROM if other than D).
4. From the pre-installer menu, choose the appropriate option to begin the CART installation procedure.

The installation program prompts you to select a type of setup:

- **Typical:** The Typical installation provides you with all application software, tools, documentation, and sample data files that are normally available. All components will be installed within the directory structure defined during the installation procedure.
- **Custom:** Choose the Custom installation if you would like to choose specific components available for installation. To include a particular option, click the mouse once on the desired option. Be sure that a checkmark appears in the appropriate box to ensure the item will be included as part of the installation.

By default, CART is installed in **C:\Program Files\Salford Data Mining\CART 6.0**. Each component of the CART installation is installed in a subfolder under **CART 6.0**.

Ensuring Proper Permissions

If you are installing CART on a machine that uses security permissions, please read the following note.

- ✂ You must belong to the power user group on Win-NT, Win-XP and Win-2000 to be able to run CART. This is due to the way licensing works on those platforms (the information is written to a system folder to which you must have write access).

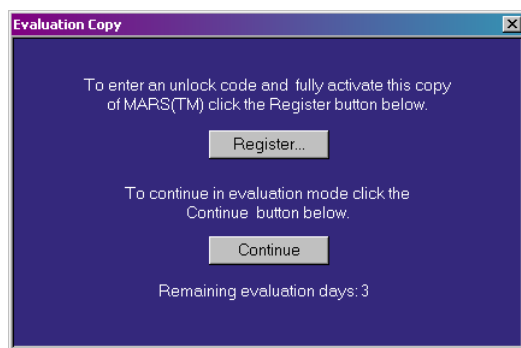
Starting and Running CART

Start CART by clicking **[Start]** and selecting the CART program group icon.

CART takes advantage of Windows preemptive multi-tasking ability, so you can start a CART run and then switch to other Windows tasks. Be aware that performance in CART and your other active applications will decrease as you open additional applications. If CART is running slowly you may want to close other applications.

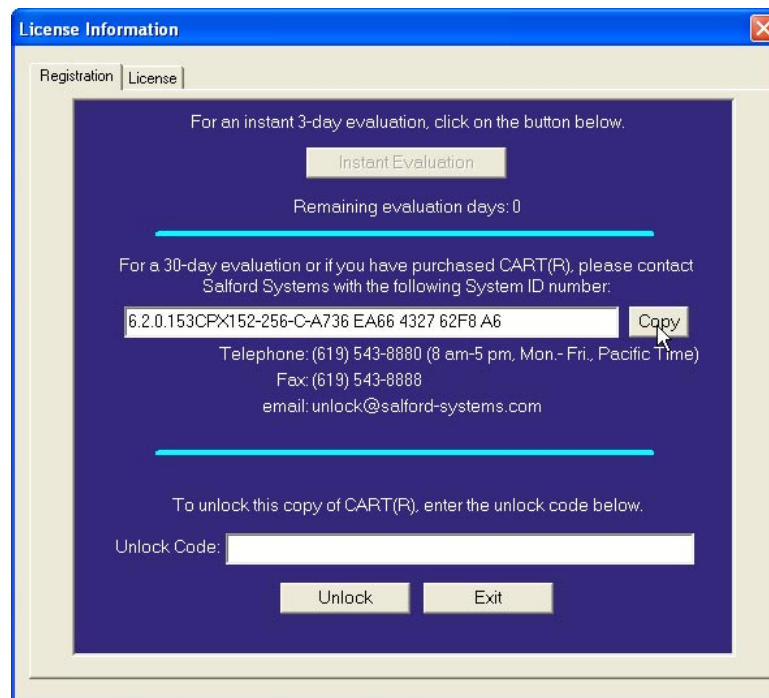
Licensing CART

After completing the install process click your start button and navigate into Program/[software]/[software], clicking on the [software] icon to start the application. You will be presented with a screen similar to the following:



Select **[Continue]** to start your instant 3-day evaluation. This will get the software up and running while you work through the unlock process. Once launched, select **License...** from the **Help** menu and choose the **Registration** tab.

Click on the **[Copy]** button to copy the System ID number



Open your email application and compose an email to unlock@salford-systems.com with the following information:

Name: Last, First

Company Name, Institution, or Affiliation

Email Address

Phone Number

System ID, which can be found by pulling down the HELP menu and then looking on the licensing information tab. (You just need to paste, Ctrl+V, from your clipboard.)

If you have not already informed us, what are you using the software for?

Once you receive the unlock code, highlight the code, right click and select **[Copy]** to copy the unlock code to your clipboard. Restart the software and go to the registration tab as you did previously and verify that the System ID number has not changed.

Place your cursor in the Unlock Code box and right click, then paste the unlock code directly into entry box. Click **[Unlock]** and you are done!



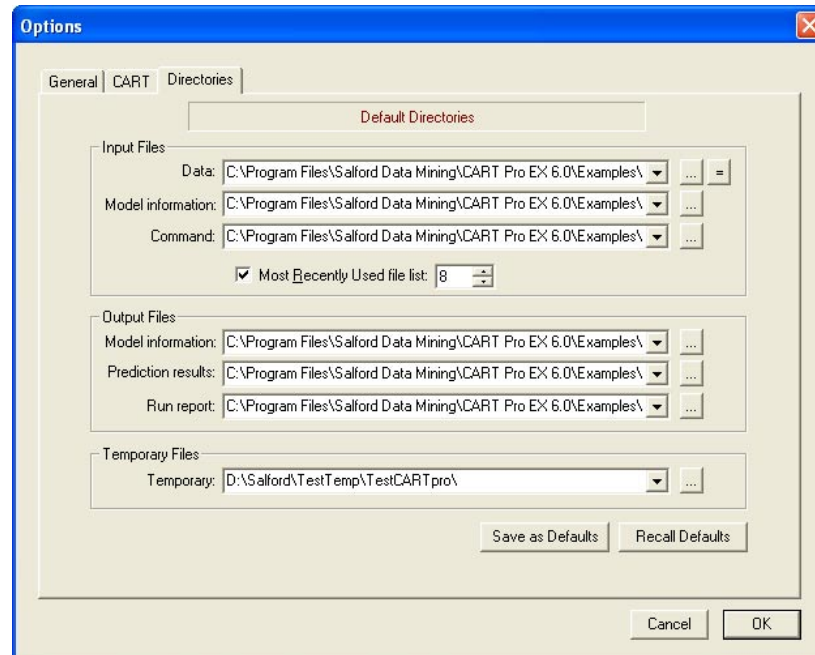
We suggest you not try to type the unlock code. A typo would invalidate the current System ID and cause the whole process to be restarted.

Preparing Your Data for CART

Accessing data for modeling and analysis. This chapter discusses file formats and rules governing ASCII and Excel files.

Setting up Working Directories

CART will utilize user-specified directories for different input and output files. First choose **Edit—Options**, then select the **Directories** tab to access/change the default locations. The tab appears as follows:



Input Files Location

- Data: –input or training data sets for modeling
- Model information: –previously-saved CART model files to be used for scoring
- Command: –command files or scripts

Output Files Location






- Model information: –CART model files saved for later scoring or export
- Prediction results: –output data sets containing scores or predictions
- Run report: –classic plain text output

Temporary Files

Temporary: –location where CART will create additional temporary files as needed

- Make sure that the drive where the temporary folder is located will have enough space (at least the size of the largest data set you are planning to use).
- ✂ Depending on your preferences, you may choose one of two working styles:
 - (1) using the same location for input and output files
 - (2) using separate locations for input and output files
- ✂ Temporary files with names like CART0314114746_.txt are records of your previous sessions. The first part of the name refers to today's date (03/14) followed by a random series of digits to give the file a unique name. These command logs provide a record of what you were doing during any session and will be stored even if you experience an operating system crash or power outage. You may find the record invaluable if you ever need to reconstruct work you were doing.
- ✂ Temporary files with names *other than* CARTnnnnn.txt are normally deleted when you shut CART down. If you find such files in your temporary directory you should delete them as they contain no useful information.

Additional Control Functions

-  –Control icon that automatically changes all path references to make them identical with the **Data:** entry.
-  –Control icon that starts the Select Default Directory dialog, allowing the user to browse for the desired directory.
-  –Control icon that automatically changes all path references to make them identical with the **Data:** entry.
-  –Control that allows you to select from a list of previously-used directories.
- ☒ Most Recently Used file list: 
 - Control that allows the user to specify how many files to show in the MRU list displayed in the **File** menu. The maximum allowable is **20** files.

Chapter

2



Reading Data

This chapter covers typical situations you may encounter while accessing your data in CART

General Comments.

The following requirements must be met to read your data successfully in CART:

- ◆ Data must be organized into a “flat file” with rows for observations (cases) and columns for variables (features).
 - ◆ The maximum number of cells (rows x columns) allowed in the analysis will be limited by your license.
 - ◆ The maximum number of variables allowed in the analysis is initially set to 32768. See the appendix for dealing with larger numbers of variables.
 - ◆ CART is case insensitive for variable names; all reports show variables in upper case.
 - ◆ CART supports both character and numeric variable values.
 - ◆ Variable names must not exceed 32 characters.
 - ◆ Variable names must have only letters, numbers, or underscores (spaces, %, *, &, -, \$, etc. are **NOT ALLOWED**). If characters other than letters, numbers, or underscores are encountered, CART will attempt to remedy the problem by substituting the illegal characters with underscores. The only exception is that character variables in ASCII files must end with a \$ sign (see the next section).
 - ◆ Variable names must start with a letter.
- Be especially careful to follow the variable name requirements because failure to do so may cause CART to operate improperly. When you experience difficulties reading your data, first make sure the variable names are legal.

Accessing Data from Salford Systems Tools

Many data analysts already have preferred database formats and use widely known systems such as SAS® to manage and store data. If you use a format we support then reading in data is as simple as opening the file. The Excel file format is the most challenging because Excel allows you to enter data and column headers in a free format that may conflict with most data analysis conventions. To successfully import Excel spreadsheets, be sure to follow the variable (column header) naming conventions below.

If you prefer to manage your data as plain ASCII files you will need to follow the simple rules we list below to ensure successful data import.

Reading ASCII Files

CART has the built-in capability to read various forms of delimited raw ASCII text files. This built-in capability is most appropriate for datasets composed of numeric and quoted character data, using a comma for the delimiter. Optionally, spaces, tabs or semicolons instead of commas can separate the data, although a single delimiter must be used throughout the text data file.

ASCII files must have one observation per line, with the first line containing variable names (see the necessary requirements for variable names in the previous section). As previously noted, variable names and values are usually separated using the comma (",") character. For example:

```
DPV, PRED1, CHAR2$, PRED3, CHAR4$, PRED5, PRED6, PRED7, PRED8, PRED9, PRED10, IDVAR
0, -2.32, "MALE", -3.05, "B", -0.0039, -0.32, 0.17, 0.051, -0.70, -0.0039, 1
0, -2.32, "FEMALE", -2.97, "O", 0.94, 1.59, -0.80, -1.86, -0.68, 0.940687, 2
1, -2.31, "MALE", -2.96, "H", 0.05398, 0.875059, -1.0656, 0.102, 0.35215, 0.0539858, 3
1, -2.28, "FEMALE", -2.9567, "O", -1.27, 0.83, 0.200, 0.0645709, 1.62013, -1.2781, 4
```

Character variables are indicated by either placing a '\$' at the end of the variable name (e.g., POLPARTY\$), or surrounding the character data with quotes (e.g., "REPUBLICAN"), or both.

Distinguishing Character vs. Numeric

CART uses the following assumptions to distinguish numeric variables from character variables in ASCII files:

- ◆ When a variable name ends with "\$," or if the data value is surrounded by quotes (either ' or ") on the first record, or both, it is processed as a character variable. In this case, a \$ will be added to the variable name if needed.
 - ◆ If a variable name does NOT end with "\$," or if the first record data value is NOT surrounded by quotes, the variable is treated as numeric.
- ✎ It is safest to use "\$" to indicate character fields. Quoting character fields is necessary if "\$" is not used at the end of the variable name or if the character data string contains commas (which would otherwise be construed as field separators).
- ✎ Character variables are automatically treated as discrete (categorical). Logically, this is because only numeric values can be continuous in nature.
- ◆ When a variable name does not end with a \$ sign, the variable is treated as numeric. In this case, if a character value is encountered it is automatically replaced by a missing value.

Missing Value Indicators

When a variable contains missing values, CART uses the following missing values indicator conventions.

Numeric:

Either a dot or nothing at all (e.g., comma followed by comma). In the following example records, the third variable is missing.

```
DPV$, PRED1, PRED2, PRED3
"male", 1, , 5
"female", 2, ., 6
```

Character:

Either an empty quote string (quote marks with nothing in between), or nothing at all (e.g., comma followed by comma). In the following example records, the first and fourth variables are missing.

```
DPV$, CHAR1$, PRED2, CHAR3$, PRED4
"male", "", 1, 3.5, , "Calif"
"female", , 2, 4, ' ', "Illinois"
```

Opening the Example ASCII File

A sample ASCII file SAMPLE.CSV comes as part of the CART distribution and resides in the "\Sample Data" folder.

To open SAMPLE.CSV you should:

1. Click on **File–Open> Data File...**
2. In the **Open Data File** dialog window, choose **ASCII-Delimited Text (*.csv, *.dat, *.txt)**.
3. When you double click on SAMPLE.CSV, the **Model Setup** dialog window should appear.



The Open Data File dialog lists only those files that match the selected extension in the File of type: selection box. You must select an explicit data format to activate the corresponding data access driver.

Accessing your data regardless of the original file format

CART, as well as other Salford Systems' applications, employs built-in DATABASE CONVERSION functionality to enable you to access data in over 90 file formats, including Excel, SAS®, S-Plus, Access, etc. By default, this capability is enabled during the installation procedure.

The **Open Data File** window contains a wide selection of supported data formats. Choose the corresponding data format first to see your files.

Variable Naming

Acceptable variable names have a maximum of 32 characters, must be composed of letters, numbers and underscores, and must begin with a letter.

- ✎ Spaces are not permitted when reading raw ASCII text files. When using DATABASE CONVERSION, spaces are permitted only when the selected data file format allows them. However, in most cases the space will be converted and displayed as an underscore.

Examples of acceptable and unacceptable variable names.:

AGE_1	OK
GENDER	OK
POLPARTY	OK
1WORLD	Unacceptable; leading character other than letter
%WEIGHT	Unacceptable; leading character other than letter
SOCIAL_SECURITY_NUMBER_AND_ACCOUNT	Unacceptable, too long. Variable name will be truncated to 32 characters.
SALT&PEPPER	Unacceptable, "&" not letter, number or underscore. This character will be replaced with an underscore.

Character variable names are required to end in an additional '\$,' so if a character variable name does not end with '\$' it will be added by DATABASE CONVERSION:

NAME\$

SSNUMBER\$

Numeric variables may optionally have subscripts from 0 to 99 but CART does not use them in any special way:

CREDIT(1)	OK
SCORE(99)	OK
ARRAY(0)	OK
ARRAY(100)	Unacceptable; parenthesis will be replaced with underscore.
(1)	Unacceptable; parenthesis will be replaced with underscore.
x()	Unacceptable; parenthesis will be replaced with underscore.
x(1)(2)	Unacceptable; parenthesis will be replaced with underscore.

- When using raw ASCII text input data, CART does not check for, or alter, duplicate variable names in your dataset.

Reading Excel Files

We have found that many users like to use Excel files. Excel files are easily accessible in mode 2 using DATABASE CONVERSION drivers. However, care must be exercised when doing this. Make sure that the following requirements are met:

- The Excel file must contain only a single data sheet; no charts, macros or other items are allowed.
- Currently, the Excel data format limits the number of variables to 256 and the number of records to 65535.
- The Excel file must not be currently open in Excel, otherwise the operating system will block any access to it by an external application such as CART. On some operating systems, if the Excel file was recently open in Excel, the Excel application must be closed to entirely release the file to be opened by CART.
- The first row must contain legal variable names (see the beginning of this chapter for details).
- Missing values must be represented by blank cells (no spaces or any other visible or invisible characters are allowed).
- Any cell with a character value will cause the entire column to be treated as a character variable (will show up ending in a \$ sign within the Model Setup). This situation may be difficult to notice right away, especially in large files.
- Any cell explicitly declared as a character format in Excel will automatically render the entire column as character even though the value itself might look like a number—such cases are extremely difficult to track down.

- ◆ It is best to use the cut-and-paste-values technique to replace all formulas in your spreadsheet with actual values. Formulas have sometimes been reported to cause problems with reading data correctly.
- ◆ Alternatively, you may save a copy of your Excel file as a comma-delimited file (.CSV) and use the **File of type:** Delimited Text (*.csv, *.dat, *.txt) (caution: make sure no commas are part of the data values).

CART BASICS

*This chapter provides a hands-on exercise using
a credit risk binary classification example.*

CART Tutorial

This chapter provides a hands-on tutorial to introduce you to the CART graphical user interface—menus, commands, and dialogs. See firsthand how easy CART is to use! In this first tutorial, you will learn how to set up a simple CART analysis, how to navigate the dynamic tree displays, and how to save your work.

- ✂ A word on our examples: CART can be applied to data from any subject. We have come across CART models in agriculture, banking, genetics, marketing, security, and zoology, among many others, and the citations to CART number in the thousands. Because analysts prefer to work with examples from their own fields we have included a few alternative case studies.
- ✂ This chapter deals with a simple YES/NO outcome drawn from the field of credit risk. If you prefer to work through a marketing segmentation example instead, you can jump to Chapter 11. Chapter 4 works through a biomedical example, and Chapter 5 using a discussion a housing regression tree example.
- ✂ We recommend that you try to follow this first example as it primarily uses concepts with which most readers will be familiar.

Our first tutorial file, GOODBAD.CSV, contains data on 664 borrowers, 461 of whom repaid a loan satisfactorily and 203 who defaulted. Clearly, the defaulters have been oversampled; few lenders could afford to have a loss rate as high as 31%. While the data have their origin in the real world, the specific records included here have been fictionalized. Nevertheless, we have retained the broad statistical relationships between the variables to yield a realistic study.

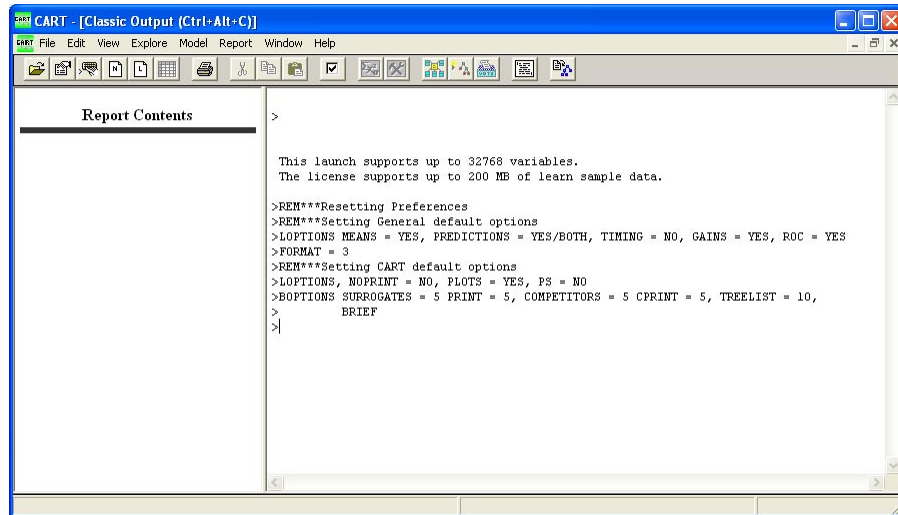
The variables available on the file include:

TARGET	0=good, 1=bad (defaulted)
AGE	Age of borrower in years
CREDIT_LIMIT	Loan amount
EDUCATION\$	Category of level of schooling attained
GENDER	Male or Female
HH_SIZE	Number of family members
INCOME	Per month
MARITAL\$	Marital status
N_INQUIRIES	Credit bureau measure
NUMCARDS	Number of credit cards
OCCUP_BLANK	No occupation listed
OWNRENT\$	Home ownership status
POSTBIN	Postal code
TIME_EMPLOYED	Years work experience

The goal of our analysis is to uncover the factors that are predictive of default. In such studies the predictors such as AGE and INCOME must pertain to the time at which the borrower was granted the loan and the TARGET records whether or not the loan was satisfactorily repaid subsequently. A successful default model could be used to create a credit score and help the lender differentiate between good and bad risks in future loan applicants.

CART Desktop

Double-click on the CART program icon and you will see a screen similar to:



Don't worry if some of the minor details are different on your screen. Later you will learn how to customize what you see when the program is started.

About CART Menus

When you first start CART you see one set of menus but the menu items will change as you progress through an analysis. Menus can change to reflect the stage of your analysis and the window you have active. As a result, not all menus are always available. Similarly, when not accessible, the commands that appear in the pull-down menus and the toolbar icons are disabled.

An overview layout of the main CART menus is presented below.

- FILE**
 - Open data set, Navigator file, Grove File, or command file
 - Save analysis results, Navigator file, Grove file, or command file
 - Open a CART notepad for creating command scripts
 - Specify printing parameters
 - Activate interactive command mode
 - Submit batch command files
- EDIT**
 - Cut, copy and paste selected text
 - Search and replace text
 - Specify colors and fonts
 - Control reporting options
 - Set random number seed
 - Specify default directories
- VIEW**
 - Open command log
 - View data
 - View descriptive statistics
 - Display next pruning
 - Assign class names and apply colors
 - View main tree and/or sub-tree rules
 - Overlay gains charts
 - Specify level of detail displayed in tree nodes
- EXPLORE**
 - Generate frequency distributions
- MODEL**
 - Specify model setup parameters
 - Grow trees/committee of experts
 - Generate predictions/score data
 - Translate models into SAS®, C, PMML, or Java
- TREE**
 - Prune/grow tree one level
 - View optimal/minimum cost/maximal tree
 - View tree summary reports
- REPORT**
 - Control CART reporting facility
 - Advanced HotSpot and TTC reports (*featured in ProEX*)
- WINDOW**
 - Control various windows on the CART desktop
- HELP**
 - Access online help

About CART Toolbar Icons

The commands used most commonly have corresponding toolbar icons. Use the following icons as shortcuts for:

















Open a data file



Submit a command file or stored script



Turning command-line entry mode on or off


-  Opening the command log to view your session history
-  View Data File
-  Print the active window
-  Cut selected text to clipboard
-  Copy selected text to clipboard
-  Paste clipboard text
-  Set major reporting options, and working directory locations
-  Display statistics for current data
-  Open activity window
-  Model Setup
-  Grow a tree or launch an analysis
-  Grow an Ensemble or Committee of Experts model
-  Translate a model into computer code
-  Score data (use a model to make predictions)

Keyboard Shortcuts

The standard Windows keyboard conventions can also be used to activate menu selections. For example, pressing **<ALT+F>** will activate the **F**ile menu because “F” in the **F**ile menu is underlined. You can also use the keyboard to activate frequently-used menu commands. The keyboard equivalents for these commands appear on the pull-down menus after the command names.

Opening a File

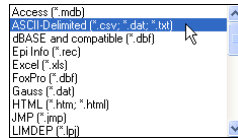
To open the GOODBAD.CSV file:

Select Open->Data File... from the File menu (or click on the  toolbar icon).

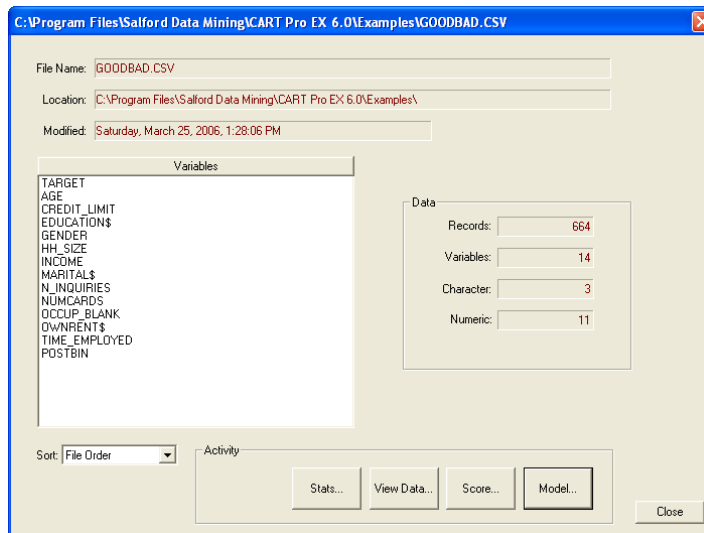
✎ Note that you can reset default input and output directories; select **Options...** from the **E**dit menu and select the **D**irectories tab.

In the **Open Data File** dialog, first navigate to the CART 6.0 Sample Data directory and then select the GOODBAD.CSV file and click on **[Open]** or double-click the file name.

As illustrated below, **Delimited Text (*.csv, *.dat, *.txt)** must be selected in the **Files of Type:** box to see files ending with the .CSV extension.

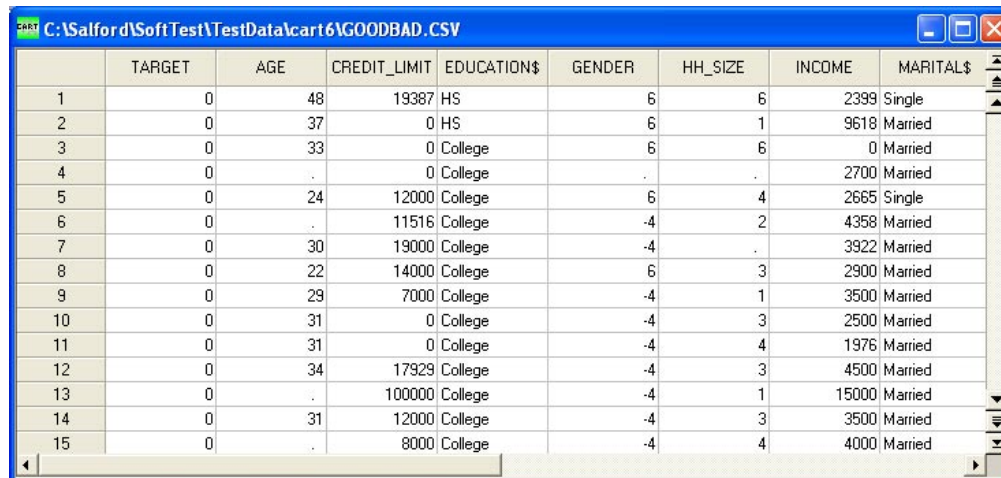


You may see a slightly different list of files in your directory. When you open GOODBAD, the **Activity** dialog opens automatically, as shown next.




We can see from here that our file contains 664 records and 14 variables, three of which are character, or text, columns. The variable names are also listed and you can change the order they are sorted in from Alphabetical to File Order using the **Sort:** drop-down control.

Start by clicking on the **[View Data...]** button to bring up a spreadsheet display of the file contents. Note that some of the cells are blank or contain only a "."; these are missing values. The window offers a view-only display; you can scroll through the data but you cannot edit it from here.





	TARGET	AGE	CREDIT_LIMIT	EDUCATION\$	GENDER	HH_SIZE	INCOME	MARITAL\$
1	0	48	19387	HS	6	6	2399	Single
2	0	37	0	HS	6	1	9618	Married
3	0	33	0	College	6	6	0	Married
4	0	.	0	College	.	.	2700	Married
5	0	24	12000	College	6	4	2665	Single
6	0	.	11516	College	-4	2	4358	Married
7	0	30	19000	College	-4	.	3922	Married
8	0	22	14000	College	6	3	2900	Married
9	0	29	7000	College	-4	1	3500	Married
10	0	31	0	College	-4	3	2500	Married
11	0	31	0	College	-4	4	1976	Married
12	0	34	17929	College	-4	3	4500	Married
13	0	.	100000	College	-4	1	15000	Married
14	0	31	12000	College	-4	3	3500	Married
15	0	.	8000	College	-4	4	4000	Married

Closing the View Data window puts us back in the Classic Output, so we click on the Activity Window icon  and select the **Model Setup** toolbar icon to reach the Model Setup dialog.


Setting Up the Model

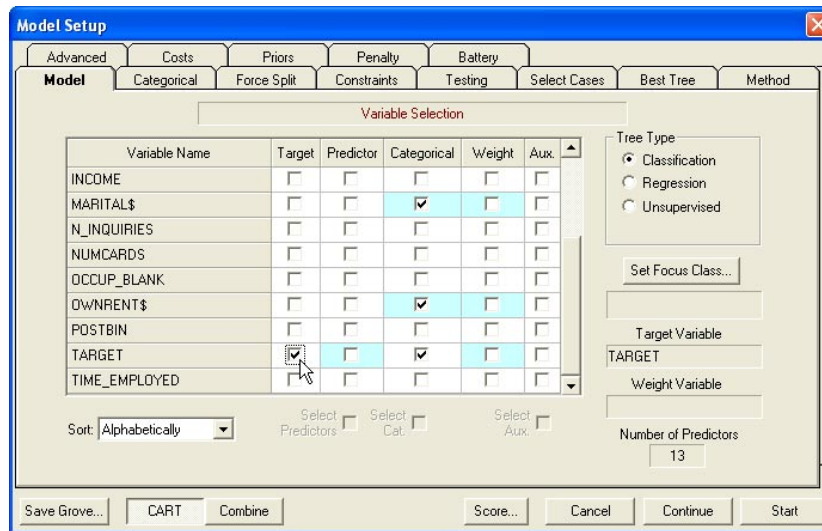
The Model Setup dialog tabs are the primary controls for conducting CART analyses. Fortunately you only need to visit the first **Model** tab to get started so we now focus on this one tab.

 Tab headings are displayed in **RED** when the tab requires information from you before a model can be built.

 In our example, the tab is red because we have not yet selected a **TARGET** variable. Without this information CART does not know which of the 14 variables we are trying to analyze or predict. This is the only **required** step in setting up a model. Everything else is optional.

Selecting Target and Predictor Variables

For this analysis, the binary categorical variable TARGET (coded 0/1) is the target (or dependent) variable. To mark the target variable, use the  to scroll down the variable list until the TARGET name is visible and place a checkmark as shown below.

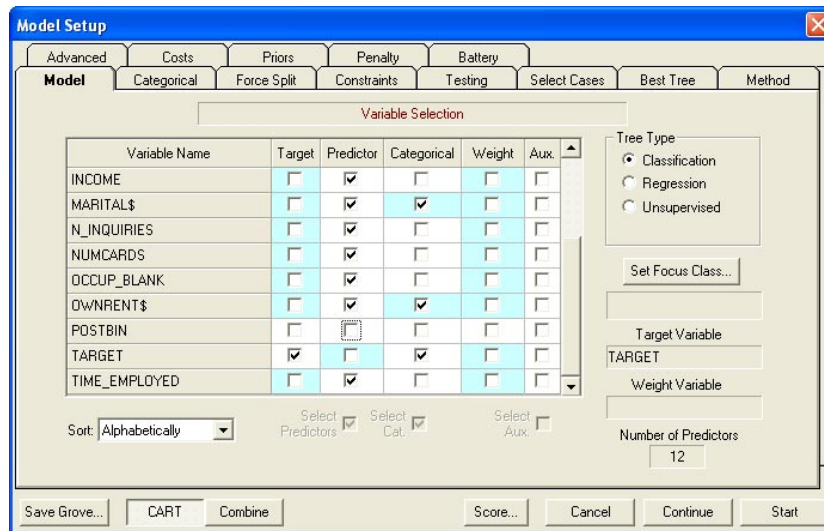


- ✂ To be safe it is also worth placing a check mark in the Categorical column. Although CART typically assumes that you intend to conduct a classification and not a regression analysis it is wise to remove any possibility of doubt.

Next we indicate which variables are to be used as predictors. CART is a capable automatic variable selector so you do not have to do any selection at all, but in many circumstances you will want to exclude certain variables from the model.

- ✂ If you do not explicitly select the predictors CART is allowed to use, then CART will screen all variables for potential inclusion in its model.
- ✂ Even if all the variables available are reasonable candidates for model inclusion it can still be useful to focus on a subset for exploratory analyses.

In our first run we will select all the variables **except** POSTBIN. Do this by clicking on the **Predictor** column heading to highlight the column, check the **Select Predictors** box underneath the column and then uncheck POSTBIN. Your screen should now look something like:



Categorical Predictors

In this data set TARGET is a categorical variable and should be checked as such. The other categorical variables, such as **MARITAL\$**, have been automatically checked as categorical predictors because they are character (text) variables.

ADD SENTENCE ABOUT NON-CHARACTER CATEGORICALS??

Growing the Tree

To prepare for model building we only need to follow these three simple steps:

- Open a file for analysis
- Select a target variable
- Indicate which numeric variables, if any, should be treated as categorical

In this case we also decided not to use one variable in the analysis. We are now ready to grow our tree.

To begin the CART analysis, click the **[Start]** button. While the model is being built a progress report will keep you informed about the actions being taken and some timing information (time elapsed, time remaining). Our example will run so fast you may not have a chance to notice everything on the progress indicator.

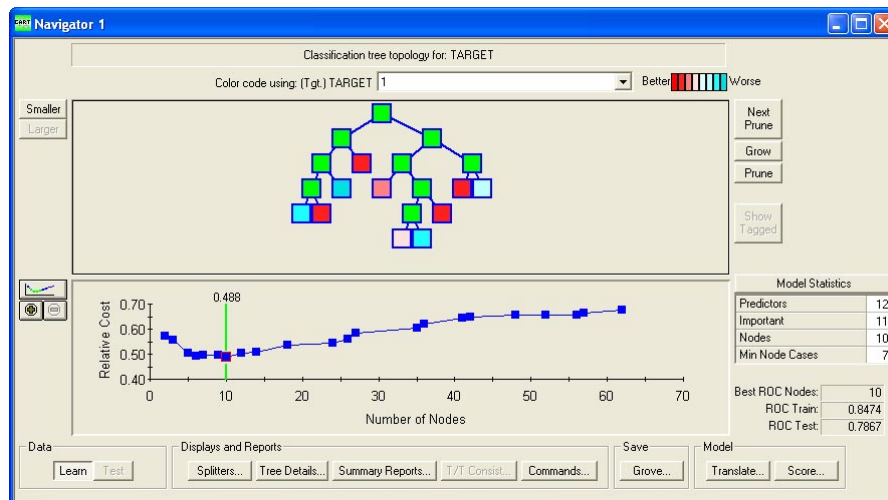
Once the analysis is complete, a new window, the **Navigator**, is opened. The navigator is the key to almost all CART output, reports and diagnostics, so it will function as a model summary and guide to everything you may want to know about the results. Experts may also redirect the classic text output and some other reports elsewhere. These items are later discussed in this manual.

Tree Navigator

The navigator packages everything you need to know about the CART tree. You can save the navigator, email it to others, or just use it temporarily during a single CART session. The navigator will offer you many views of the model and its findings, will allow you to score new data, and can generate formatted text reports, tables, charts, and comparative performance summaries of competing models. The rest of this chapter is devoted to discovering the charms of the navigator.

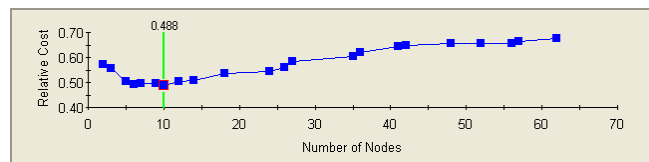
The initial navigator display is just a simple overview of the shape of the tree or its topology in the top panel, and a predictive performance curve in the bottom panel. The tree topology, displayed in the top panel of the Navigator window, provides an immediate snapshot of the tree's size and depth. Here we have a tree with 10 terminal nodes (nodes at the bottom of the tree).

The color-coding helps us locate interesting terminal nodes. Bright red nodes isolate defaulters (Target class 1) and deep blue nodes are heavily populated with good borrowers. Other colors indicate more mixed results.



The tree displayed automatically is of the size determined by CART to be the most accurate classifier obtained. Other tree sizes are also available for display. In this example we can review trees with as few as two nodes or as many as 62 nodes.

The performance of the different-sized trees is displayed in the lower panel of the navigator. This curve is a relative cost profile and traces the relationship between classification errors and tree size.



We call this a relative error curve because it is always scaled to lie between 0 and 1. 0 means no error or a perfect fit, and 1 represents the performance of random guessing. The best that we can do for the current tree is indicated by the green bar marking the low point on the error profile, where we hit a relative error of .488. If we settle for either too small or too large a tree we will not do as well as we could with the 10-node tree. Here we see the characteristic U-shaped curve with a partially flattened bottom.

- ✂ At this stage all you need to keep in mind is that we are looking for trees with low values of relative error.
- ✂ A tree with a relative error of 0 or near 0 is usually too good to be true. In almost all cases this results from including an inappropriate predictor in the model.
- ✂ It is possible to have a relative error greater than 1. This happens when the model is actually worse than random guessing.

Returning to the navigator we see some core **model statistics** in the bottom right section. The report shows that we conducted the analysis with 12 predictors, of which 11 were found to have some value. The tree being displayed now has 10 terminal nodes and the smallest of these nodes contains seven records.

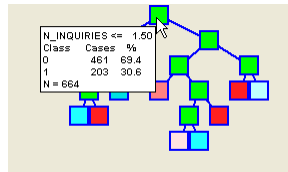
Just below the main model statistics are ROC measures. If you are not familiar with the ROC we include some introductory material on this important metric in a later chapter. For right now, all you need to know is that the ROC can range between 0 and 1 with higher values indicating better performance. Our model shows excellent performance with a test value of the ROC of .7867.

- ✂ Suppose we were to take a single good borrower and a single defaulter at random from a data set. Our ROC score tells us that we would be able to correctly tell which one was the defaulter in 78.67% of all cases.

- ✂ If you picked the defaulter at random you would be right on average for 50% of all cases. Therefore, a good model needs to deliver substantially better than an ROC of .50. In real world credit risk scoring, an ROC of .70 would be considered respectable.
- ✂ The predictive performance of a model depends on many factors, including the nature and quality of the data and the inherent predictability of the data under study. You cannot expect every subject matter to support highly accurate models.

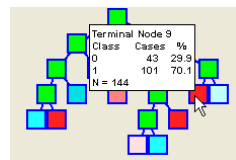
The color-coding of the terminal nodes is controlled from the pull down control at the top of the navigator. For 0/1 target variables the default coloring uses red to indicate a high concentration of 1s. You can change that if you prefer to have red represent another class instead, and you can also turn off special color coding, leaving all the terminal nodes red.

CART offers many ways to view the tree details and interior. We will start by hovering the mouse over a node. Beginning with the root node at the top of the tree, we note that we started with 461 GOODs (0s) and 203 BADs (1s), for a bad rate of 30.6.



- ✂ You can change the detail revealed when you hover your mouse over navigator nodes. Right-mouse-click in the “gray” area of the navigator window to bring up the patterns available, then left-mouse-click on your preferred display. You can also use **View->Node Display** menu to control mouse hover displays.

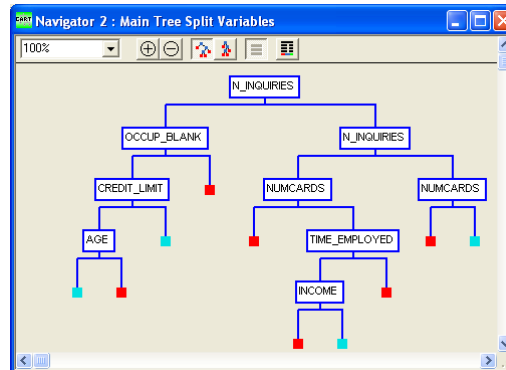
Now hover over the bright red node near the bottom right of the tree. This is terminal node 9, which has a bad rate of 70.1%, substantially higher than the baseline rate of 30.6% in the root. Visiting the other bright red nodes reveals similarly concentrated groups of defaulters.



Having established that our tree appears to be a promising model we now want to drill deeper into the results.

Viewing the Main Splitters

A convenient way to get a bird's eye view of the model is to reveal only the variables used in each node. At the bottom left of the navigator click on the **[Splitters...]** button to see:



The color coding here is a simplified one: red means “above average” risk and blue means “below average risk.” Because the CART tree splitters always send low values of a splitter to the left and high values to the right, reading this display is easy. Going down the right side of the tree we see that if a person has a large number of inquiries but few credit cards they are quite high risk. Presumably this means that the person has probably attempted to obtain additional cards in the recent past but has failed. Looking down the left-hand side of the tree we see that persons who have a low number of inquiries but did not report an occupation are also high risk.

✂ Remember that these data are fictionalized and so should not be thought of as a completely faithful representation of real world credit risk. Some surprises are inevitable in this example.

We find the splitters view of the tree helpful in giving us a quick overview of the main drivers in the tree. We see the variables used at the top of the tree and the direction of their effect. At the bottom left we see that being older is a default risk factor and at the bottom middle we see that a lower income is also a risk factor. These are just quick impressions that help us acquire a feel for the message of the tree.

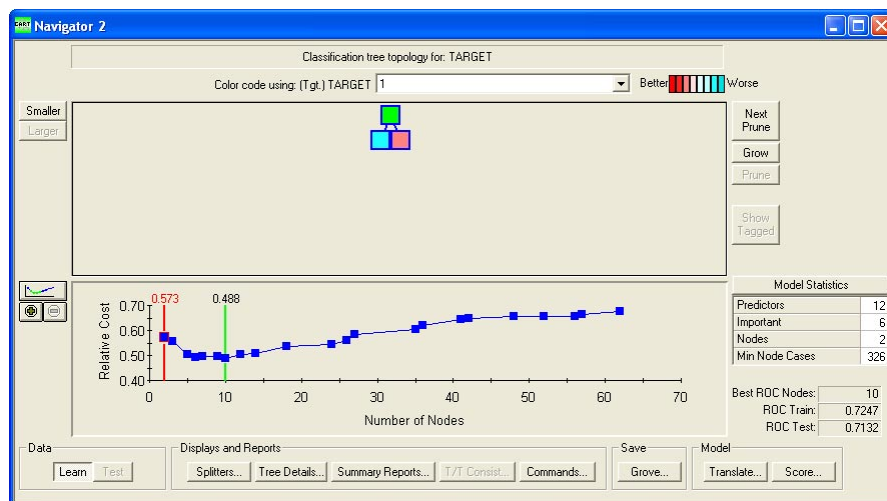
✂ The splitters view is an excellent way to quickly detect significant data errors. If you see a pattern of outcomes that is very different from what is expected or even possible you have identified a potential data flaw that needs to be investigated.

Exploring Trees of Different Sizes

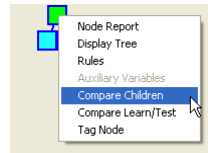
When a CART run completes, it displays the CART “optimal” tree: typically the tree with the smallest misclassification rate (or equivalently the highest classification accuracy). There are reasons to want to look at trees of different sizes, however:

- ♦ The relative error profile is often flat near its minimum. This means that smaller trees exist that are almost as accurate as the best tree found.
- ♦ Classification accuracy is not the only sensible criterion to use to select a model. Many data mining specialists prefer to use the area under the ROC curve as their model selection criterion.
- ♦ For decision making purposes you may be interested only in the top-performing nodes of the tree. If so, the accuracy and reliability of these nodes are all that matter and the overall performance of the tree is not relevant.
- ♦ Judgment can play an important role in the final tree selection.

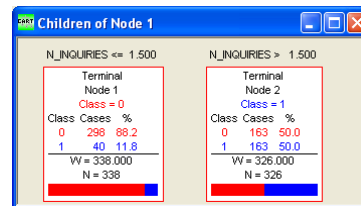
The navigator makes it very easy to view, display, and obtain reports for every size of tree found by CART in its tree-building process. Select the navigator window and then use your left and right arrow keys to display different-sized trees in the navigator topology display. Begin by moving all the way to the left to reach the two-node tree:



Technically we could go one step further to arrive at the one-node tree (the null tree), but we make the two-node tree the smallest we will display. This tree makes use of only one predictor and is actually quite predictive, with a relative error rate of .573 and a test sample ROC value of .7132. This is unusually good for a single predictor and is far from typical. To take a closer look, move your mouse over the root and **right-click** to reveal this menu:



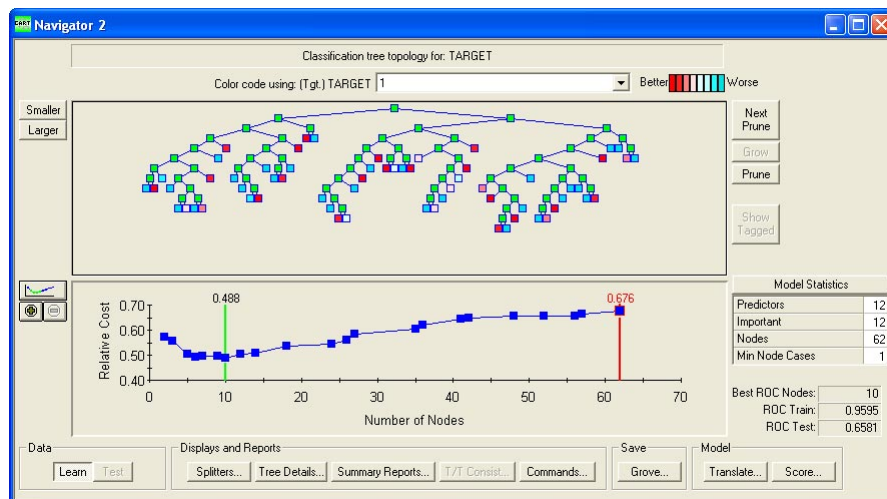
Select **Compare Children** to get the following display:



We see that having had more than one recent inquiry about a borrower at the credit bureau is a powerful indicator of default risk. Recall that the default rate in these data is 30.6% overall, whereas it is only 11.8% among those with one or no recent inquiries and 50% for those with two or more recent inquiries. (You can customize the colors and details shown in this window using the **ViewNode Detail...** menu discussed later.)

- ✎ CART trees are grown by a procedure called “binary recursive partitioning.” The **binary** indicates that when a node is split it is divided into two and only two child nodes. This is a distinctive characteristic of the CART tree and is one source of the power of the CART technology.
- ✎ CART easily creates the equivalent of multi-way splits by using a variable more than once. We show an example below.

Close the “Children of Node 1” window and use the right-arrow key to move all the way to the other extreme: the largest tree grown in this run. From the relative error profile and the model statistics you can see that this tree has 62 nodes, its relative error is .676, and the test ROC is .6581.



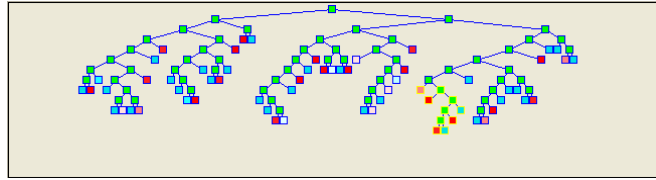
This largest tree is quite a bit worse than the simple two-node tree, indicating that the large tree is seriously “overfit.” While the largest tree is almost always overfit it is not necessarily worse than the smallest tree. In some cases the largest tree is also quite accurate, though in this example it is not.

The largest tree is actually the starting point for CART analysis. CART first splits the root node, then splits the resulting children, then splits the grandchildren, and so on. The CART tree does not stop until it literally runs out of data. This is in contrast to other decision trees that use a “stopping rule.”

✂ The CART approach to decision tree construction is based on the foundation that it is impossible to know for sure when to stop growing a decision tree. (You can prove this mathematically.) Therefore, CART does not stop, but rather grows and grows and grows.

✂ CART uses extraordinarily fast proprietary algorithms so it does not take much time to grow the initial largest tree.

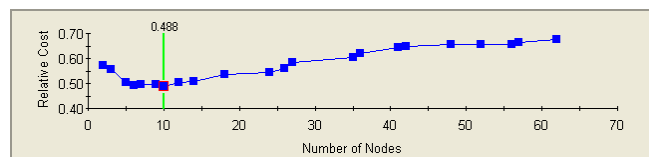
Once we have the largest tree constructed we begin **pruning**. (This is done for you automatically.) The pruning process trims the tree by removing the splits and branches that are least useful. A pruning step often removes just one split but sometimes several splits are removed together. (The mathematical details are provided in the original CART monograph.) To see which nodes are removed in the next pruning step, click on the **[Next Prune]** button at the upper right side of the navigator. The nodes to be pruned next will be highlighted in yellow. Use the left arrow key to return to the CART optimal tree marked with the green bar.




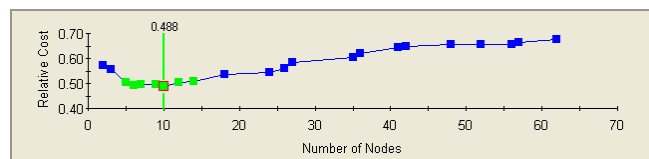
✂ The **Home** key is a short cut to return to the CART optimal tree in the navigator.

Here we can clearly see which node would be pruned next if we wanted to select a smaller tree. The reason CART would prune this particular node next is that by doing so CART would retain as much accuracy as possible. Now click on Next Prune again to turn off the node highlighting.

Look again at the relative error profile and note the flat region near the 10-node mark.



It is natural to suspect that one of these smaller trees is practically just as good as the optimal tree. If you click on the  on the left side of the navigator you will see a portion of the relative error profile turn green.



This tells us exactly which sizes of trees exhibit an accuracy performance that is statistically indistinguishable from the optimal tree. The CART authors suggested that we use a “1 standard error” or 1SE rule to identify these trees and in the display we have moved to the smallest of these trees.

✂ The 1SE tree is the smallest tree displaying an error rate that is no worse than one standard error above the optimal tree.

✂ Because determining which tree is actually best is subject to statistical error we cannot be absolutely certain which tree is best. Every tree marked in green is a defensible candidate for “best tree.”

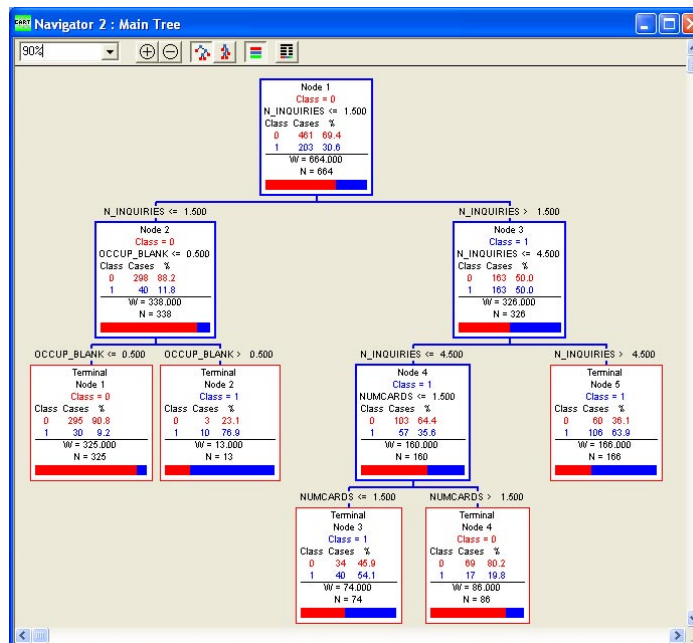
✂ In our example the 1SE tree has five terminal nodes, with a relative error of .504 and a test ROC of .7552. The optimal tree has a relative error of .488 and a test ROC of .7867. The optimal tree is “better” but it is also twice the size and our measurements are always subject to some statistical uncertainty. For the next displays we will work with the 1SE tree.

A tree of a specific size can be selected in several ways:

- ♦ use the mouse to click on a blue box in the error profile
- ♦ use the left and right arrow keys to reach a specific tree
- ♦ click the **[Grow]** or **[Prune]** buttons on the right side of the navigator
- ♦ from the **T**ree menu select a tree or list of trees







Viewing the Main Tree

The **[Tree Details...]** button on the navigator brings up an industry standard view of a decision tree. This view includes node-specific sample breakdowns so that we can see performance throughout the tree. Starting with the five-node 1SE tree selected, click on the **[Tree Details...]** button at the bottom of the Navigator (or right-click on the root node and select the **Display Tree** option) to get:



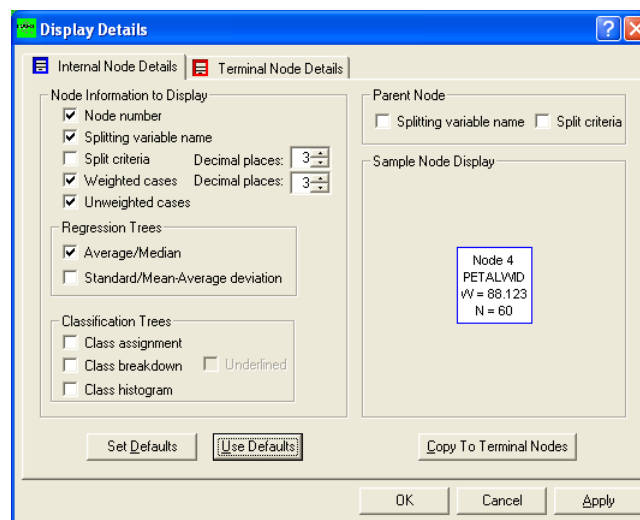
The example shows how CART creates multi-way splits from binary split building blocks. The root node first splits on $N_INQUIRIES > 1.5$ and then again on $N_INQUIRIES > 4.5$. This creates three regions for $N_INQUIRIES$: {0 or 1}, {2, 3, or 4}, and {5 or more}.

With a mouse click you can:

- ◆ Zoom in or Zoom out by pressing the  or  keys
- ◆ Fine-tune the scale by changing the  selection box
- ◆ Experiment with two alternative node-spacing modes ( and  buttons)
- ◆ Turn color coding of target classes on or off ( button)

Try clicking on these controls now to see what they do.

The detail appearing in each of the nodes can be customized separately for internal and terminal nodes. From the **View** menu, select **Node Detail...**; the following dialog appears:



The default display setting is shown in a sample node in the right panel. Click on the check boxes to turn each option on and off and then click **[Apply]** to update the Main Tree display. To save your preferred display options as the default settings, click the **[Set Defaults]** button.

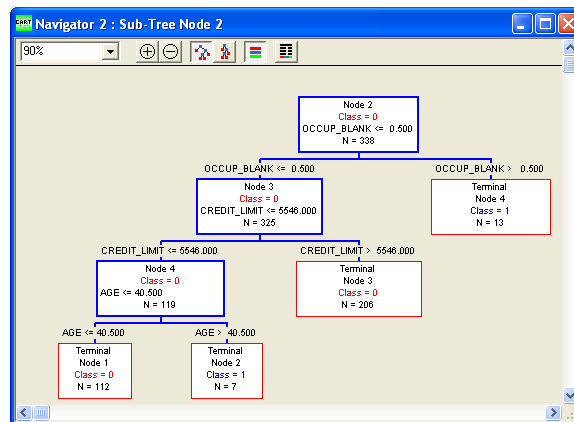
The internal and terminal node detail can be specified separately as each is given its own tab. Press the **[Copy to Terminal Nodes]** or **[Copy to Internal Nodes]** buttons if you wish the current setup to be copied into the other tab.

🔧 The **[Set Defaults]** button only sets the defaults for the currently active tab. If you want to set defaults for both terminal and internal nodes, press this button twice, once for each tab.

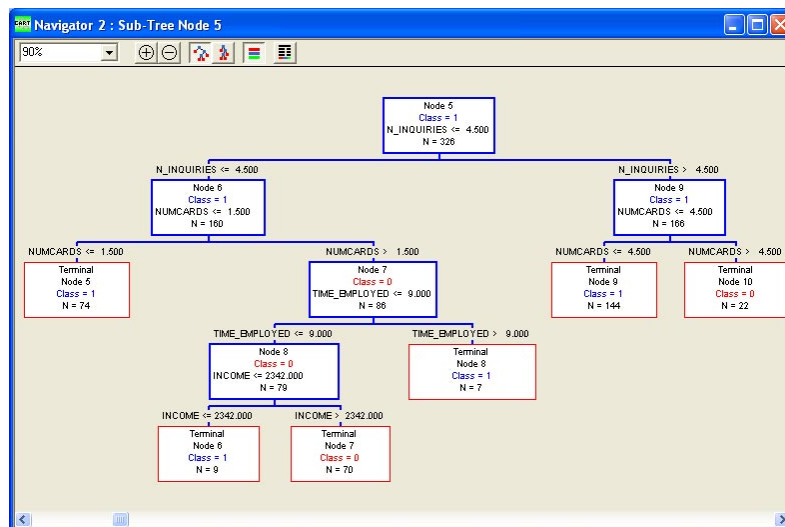
Viewing Sub-trees

Sometimes the tree you want to examine closely is too large to display comfortably on a single screen, and looking at a sub-tree is more convenient. Sometimes you will want to look at two separated parts of the tree side by side. To view sub-trees, first go back to the navigator (you can close the tree details window or select the navigator from the **Window** menu).

Next, right-click on an internal node, and select **Display Tree**. Below we have done this twice: once for the right child of the root and again for the left child, bringing up two sub-tree displays. Below we display the two windows side by side.



Left Child Node



Right Child Node

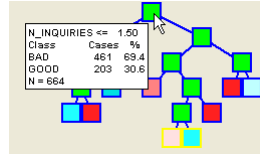
Assigning Labels and Color Codes

Trees detailing sample breakdowns can be displayed with or without colors; the node histograms are always color-coded. Instructions for customizing the colors appear below. If your target variable is coded as text then the text value labels will be displayed where required, but if your target variable is coded as a number you can replace the numbers with labels with **Class Names**.

Class names (up to 32-characters) and colors can be assigned to each level of the target variable from **View** menu:

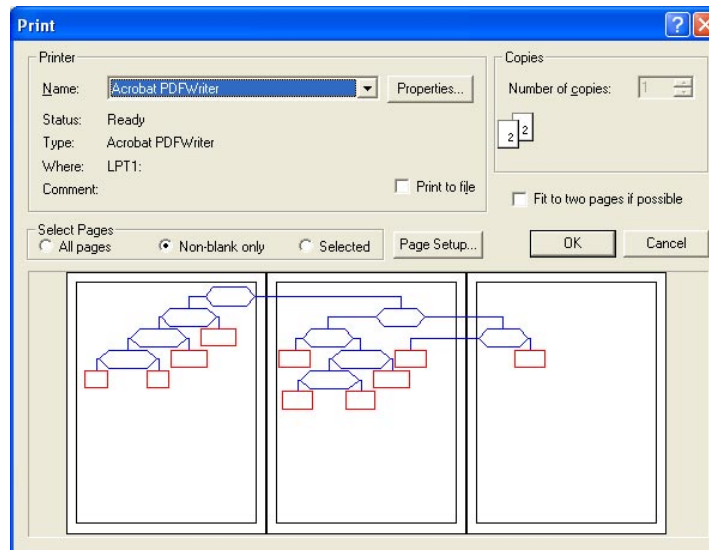
1. Select **Assign Class Names...**
2. Click on the **Name** text box and enter a label for that class.
3. Click on **[Color...]** to select a color from the palette, then click **[OK]**.
4. Click **[Apply]** to enter the name/color; repeat steps 2-4 for the other levels.

An illustrative Class Assignment dialog box for our example is shown below. The labels and color codes are displayed in the individual node detail you see when you hover the mouse pointer over a node in the **Navigator** window, as well as in the main and sub-tree diagrams and printed tree output.



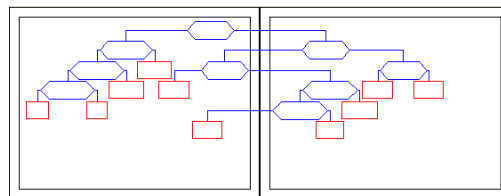
Printing the Main Tree

To print the Main Tree, bring the tree window to the foreground and then select **Print** from the **File** menu (or use **<Ctrl+P>**). In the Print dialog box, illustrated below, you can select the pages that will be printed and the number of copies, as well as specify printer properties. You can also preview the page layout; CART will automatically shift the positions of the nodes so they are not split by page breaks.



You can see from the preview that a small section of the GOODBAD main tree spills over to a second and third page. To resize and reorient the tree, click on the **[Page Setup...]** button.

By selecting the **Landscape** orientation we now manage to fit the tree on two pages.

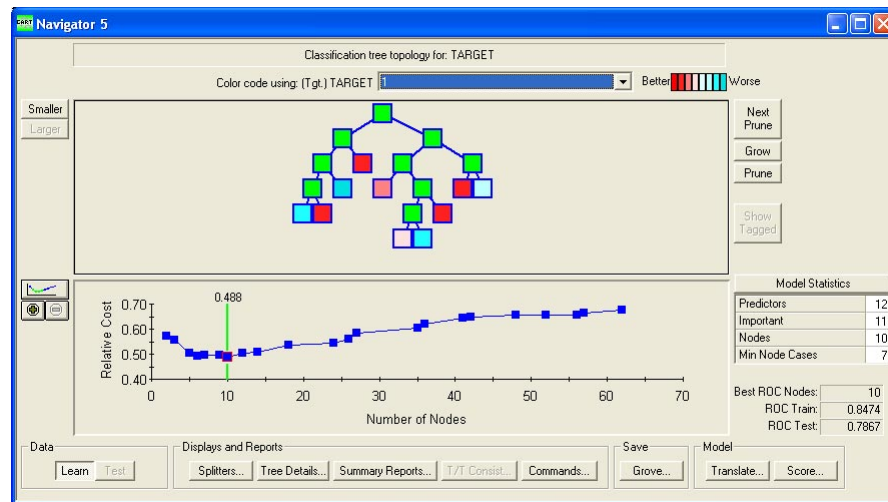


The **[Page Setup...]** is most useful with larger trees because a little tweaking can reduce the total page count dramatically. You can often obtain convenient thumbnail displays of the most complex tree by selecting **Fit to two pages if possible** on the Print menu.

Tree Summary Reports

The overall performance of the current tree is summarized in seven **Summary Reports** dialog tabs. To access the reports, click **[Summary Reports...]** at the bottom of the **Navigator** window (or select **Tree Summary Reports...** from the **Tree** menu).

Tree Summary Reports present information on the currently-selected tree, i.e., the tree displayed in the top panel of the Navigator. To view summary reports for another size of tree, you must first select that tree in the navigator. For the summary reports that follow, we work with the CART optimal tree with 10 nodes.

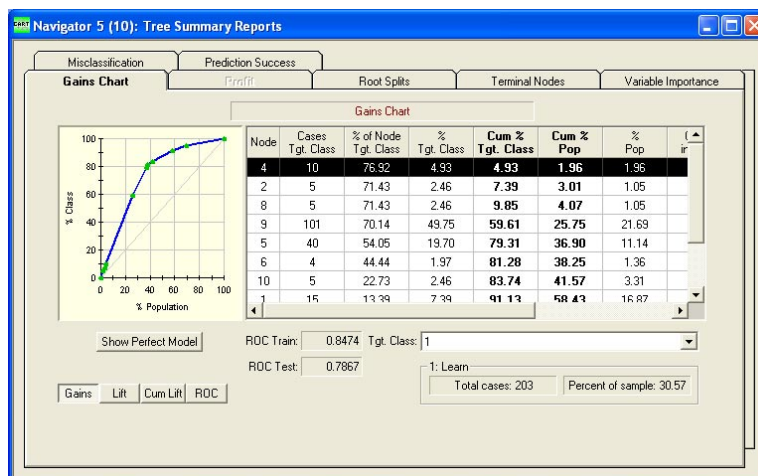


As illustrated below, the **Summary Reports** dialog contains gains charts, terminal node counts, variable importance measures, misclassification tables, and prediction success tables, as well as a report on the root node splitters and a "Profit" tab.

Gains Chart/Cumulative Accuracy Profile

The summary report initially displayed is the **Gains Chart** tab, also known in credit risk as the Cumulative Accuracy Profile (CAP) chart. Gains charts are always tied to a specific level of the target variable, which we also call the **Focus class**. If your Gains chart appears with the wrong focus class, just select the one you want from the

pull down menu in the lower right portion of the tab. Because we assigned class names, the class we are interested in is now listed as BAD instead of 1.



Reading the gains curve is straightforward. Consider the data sorted in order from most likely to be BAD to least likely. If we were to look only at the top 10% of the data (most likely to be BAD) what fraction of all the BADs would we capture? Looking at the graph it appears that we would capture about 23% of all BADs. The ratio 23/10 or 2.3 is known as the lift among market researchers and relative risk in the biomedical world. Clearly, the larger the lift the better because it indicates more precise discrimination.

✂ Click on **Show Perfect Model** to provide a reference to compare against. The perfect model would isolate all the BAD cases into their own nodes.

Our example has been run using the self-testing cross-validation method. Cross validation is a clever technique for testing models without formally dividing the data into two separate learn and test portions. However, if in your own analyses you use a test sample, buttons for selecting results based on the **[Learn]**, **[Test]**, or **[Both]** samples will appear in the lower portion of the Gains Chart dialog. To view gains charts for the test sample, click **[Test]**, and to view gains charts for learn and test combined, click **[Both]**.

✂ When you use cross validation (CV) for testing you will obtain reliable estimates of the overall classification accuracy of the tree and a test-based measure of the area *under* the ROC curve. The CV method does not produce a test-based version of the actual Gains or ROC *curve*.

✂ Because we have used CV for testing in our example we will see test results on only some of the summary tabs.

The grid displayed in the right panel contains various counts and ratios corresponding to each node of the tree and the quantities used to plot the gains curve. Remember that the nodes have always been sorted for the focus class using learn data results.

The table displays the following information for each terminal node (scroll the grid to view the last two columns):

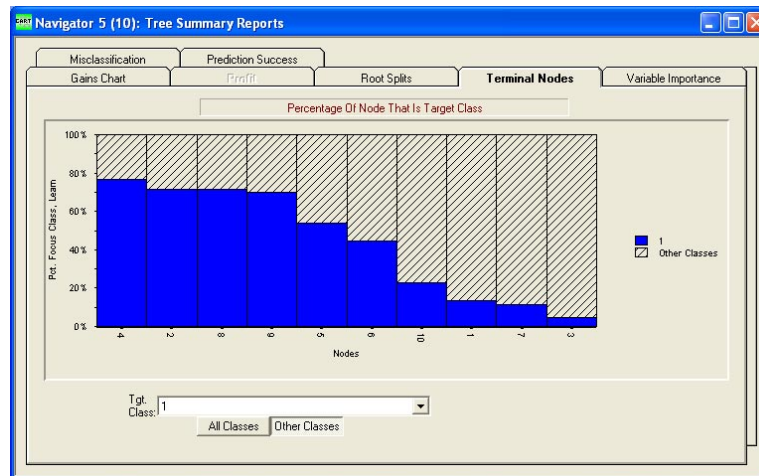
NODE	Node number
CASES TGT CLASS	N of cases in node belonging to focus class
% OF NODE TGT CLASS	Percent cases in node that are focus class
% CLASS TGT CLASS	Percent of all focus class present in node
CUM. % TGT CLASS	Cumulative percent of focus class
CUM. % POP	Cumulative percent of all data
% POP	Percent of all data in node
CASES IN NODE	N of cases in node
CUM. GAINS	Cum % Focus Class / Cum % Pop
LIFT INDEX	% node focus class/ % pop focus class

The Gains Table can be exported to Excel by a right-mouse click and then choosing **Export...** from the pop-up menu.

You can print individual Gains Charts as well as overlay and print Gains Charts for trees of different sizes and from different CART analyses (see Chapter 4). You can also add Gains Charts and Tables into the CART report (see Chapter 12).

Terminal Nodes

The next Summary Report provides a graphical representation of the ability of the tree to capture the BADs in the terminal nodes. Observe that we selected BAD as the target class. This sorts the nodes so that those with the highest concentrations of BAD are listed first. The **[All Classes]** button represents each class with its own color. The other classes are just colored gray.

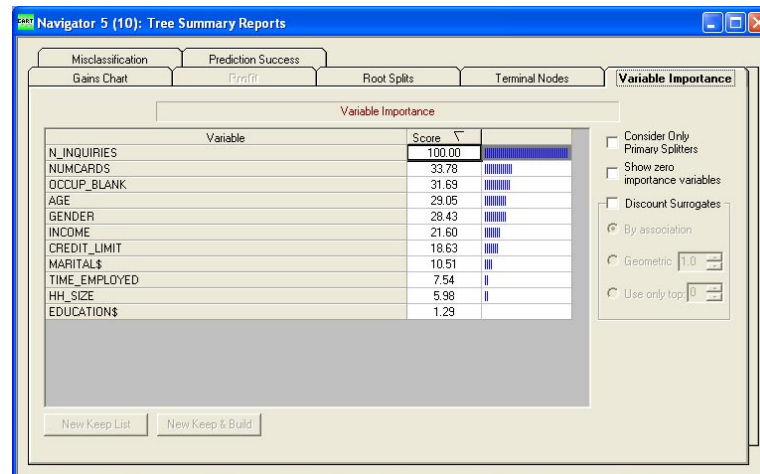


Node 4 has the highest concentration of BADs, closely followed by nodes 2, 8 and 9. Hover the mouse over a bar to see the precise fraction of the node that is BAD. This is a graphical display of the information that is also in the gains chart.

If you have separate test data you can request a learn/test comparison of the terminal nodes in this window.

Variable Importance

It is natural to expect that the root node splitter will be the most important variable in a CART tree and indeed in our example this is the case. However, you cannot count on it coming out this way in every tree. Sometimes a variable that splits the tree below the root is most important because it ends up splitting many nodes in the tree and splitting powerfully. Variable importance is determined by looking at every node in which a variable appears and taking into account how good a splitter it is. You should think of the variable importance ranking as a summary of a variable's contribution to the overall tree when all nodes are examined. The formulas for variable importance calculations are detailed in the CART monograph.

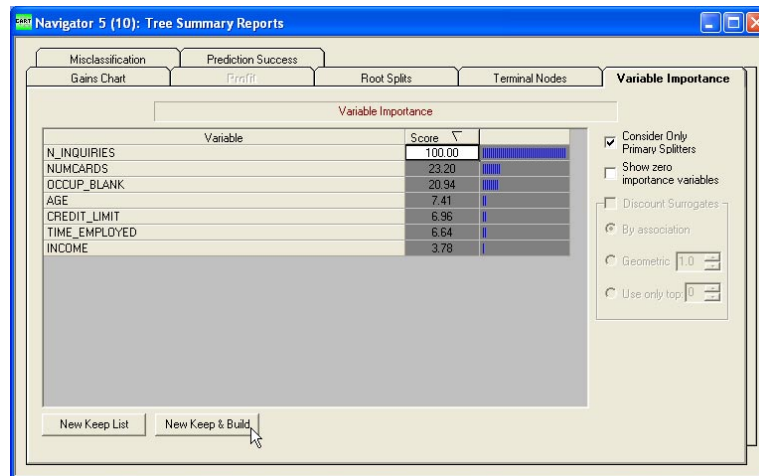


Variables earn credit towards their importance in a CART tree in two ways: as primary splitters that actually split a node, and as surrogate splitters (back-up splitters to be used when the primary splitter is missing). To see how the importance scores change if considered only as primary splitters, click the **Consider Only Primary Splitters** check box; CART automatically recalculates the scores.

✎ Comparing the standard CART variable importance rankings with the **Consider Only Primary Splitters** can be very informative. Variables that appear to be important but rarely split nodes are probably highly correlated with the primary splitters and contain very similar information.



Click inside any column of the variable importance chart to start highlighting rows. You can use this to select variables on which to focus on in a new analysis. Below we have selected the seven variables that actually appear as splitters.



- ✂ Once you have highlighted variables in this way on the variable importance chart you can automatically build a new model using only those predictors. Just click on the **New Keep & Build** button.
- ✂ Clicking on the **New Keep List** button creates a list of those variables and places them on a KEEP list in a new notepad. You can edit this KEEP command and place it in scripts or just save it for later use.

Misclassification

The Misclassification report shows how many cases were incorrectly classified in the overall tree for both learn and test (or cross-validated) samples. The tables, which can be sorted by percent error, cost or class, display:

CLASS	Class level
N CASES	Total number of cases in the class
N MISCLASSIFIED	Total number of misclassified cases in the class
PCT. ERROR	Percent of cases misclassified
COST	Fraction of cases misclassified multiplied by cost assigned for misclassification

In our example, we can see that the misclassification errors were about 19% for the learn sample and 25% for the cross-validated test results. This tab is primarily useful when working with many target classes.

CART Navigator 5 (10): Tree Summary Reports

Buttons: Gains Chart, Profit, Root Splits, Terminal Nodes, Variable Importance

Misclassification | Prediction Success

Misclassification

Learning Sample					Test Sample				
Class	N Cases	N Mis-Classified	Pct Error	Cost	Class	N Cases	N Mis-Classified	Pct Error	Cost
0	461	89	19.31	0.19	0	461	107	23.21	0.23
1	203	38	18.72	0.19	1	203	52	25.62	0.26

Sort by: Class

Prediction Success or Confusion Matrix

The confusion matrix is a standard summary for classifiers of all kinds and has been used to assess statistical models such as logistic regression as well as more exotic data mining models. We call it the Prediction Success table following Nobel Prize-winning economist Daniel McFadden's 1979 paper on the subject. The table is a simple report cross-classifying true class membership against the predictions of the model. The table for our 10-node follows:

CART Navigator 5 (10): Tree Summary Reports

Buttons: Gains Chart, Profit, Root Splits, Terminal Nodes, Variable Importance

Misclassification | **Prediction Success**

Learning Sample Prediction Success Table

Actual Class	Total Cases	Percent Correct	Predicted Class	
			0 N=410	1 N=254
0	461	80.69	372	89
1	203	81.28	38	165
Total:		664.00		
		Average:	80.99	
		Overall % Correct:	80.87	

Learn Test Class: None

Count Row % Column %

The rows of the table represent the true class and the columns the predicted class and can report either train or test sample results. Here we have chosen to display test results based on cross validation. Via cross validation we determine that for the 203 actual BADs we classify 151 of them correctly (74.38%) and 52 incorrectly.

Among the 461 GOODs we classify 354 correctly (76.79%) and 107 incorrectly. The overall % correct is simply the total number classified correctly (151 + 354) divided by 664, the total number of cases. The average % correct is the simple average of the % correct in each class (74.38% and 76.79%). In this example the two averages are very close but they may well be quite different in other models.

To export the table as an Excel spreadsheet or copy it to the CART report document just right-click anywhere in the display. As you can see from the window, you can opt to see Learn or Test results. The cells of the table in either case can contain counts, row percents or column percents.

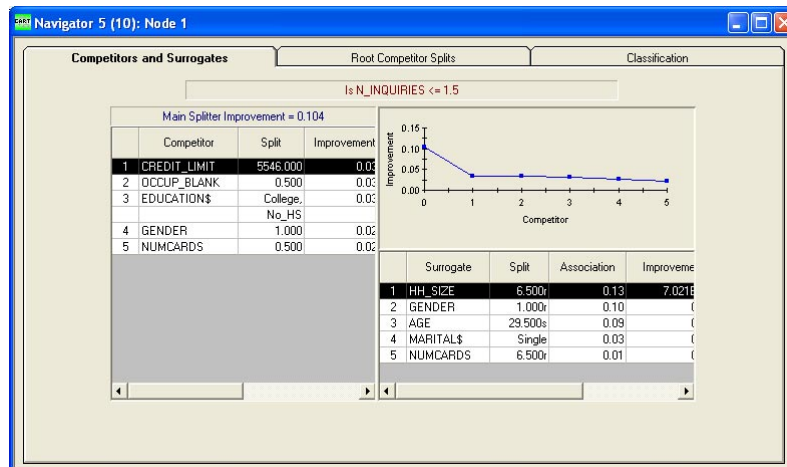
✎ Prediction success tables based on the learn sample are usually too optimistic. You should always use prediction success tables based on the test (or on cross validation, when a separate test sample is not available) as fair estimates of CART performance.

Detailed Node Reports


To see what else we can learn about our CART trees, return to the Navigator by closing the Summary Reports window or by selecting **Navigator** from the **Window** menu. Move the mouse pointer to the root (top) node in the tree topology panel and click to activate a non-terminal Node Report dialog (or right-click on the root node and select **Node Report**).

The Competitors and Surrogates tab

As illustrated below, the first of the three tabs in the non-terminal node report provides node-specific information for both the competitor and the surrogate splits for the selected node (in this case, the root node).



The splitting rule, $Is\ N_INQUIRIES \leq 1.5$, is displayed in the top line, and the **main splitter improvement** is displayed in the following line on the left. Splitter improvement is the metric CART uses to evaluate the quality of all splits; it is computed differently for different splitting rules. A table of the top five competitor splits in decreasing order of importance is displayed in the left panel. Each competitor is identified by a variable name, the value at which the split would be made, and the improvement yielded by the split.

 You may need to alter the width of the columns in this display to make everything we discuss here visible. Just position your mouse in the column header and over the border you wish to move. When the cursor changes to a cross-hairs right-click and drag the border to widen or narrow the column.

The best competitor, CREDIT_LIMIT, would split at the value 5546 and would yield an improvement of 0.0346, quite a bit below the main splitter improvement of 0.1035. Improvement scores should be looked at in relative rather than absolute terms. The improvement of the main splitter is almost three times that of the best competitor, an unusually large (but not suspiciously large) ratio. The quality of the competitor splits relative to the primary split can also be evaluated by inspecting the line graph displayed in the upper-right panel. The improvement yielded by each competitor split appears on the y-axis and the number or rank of the competitor split on the x-axis, with the primary split improvement displayed at $x=0$. The graph makes plain that the primary splitter is quite a bit better than the closest competitor but that the 2nd, 3rd, and 4th competitors all have similar improvements.

Surrogates are an important innovation in CART technology and play a key role in CART prediction and tree interpretation. A surrogate splitter is a splitter that is “similar to” the main splitter in how it assigns cases in a node to the left and right children. The top surrogate is the splitter that comes closest to matching the main splitter’s left-right assignments, but “closest” does not necessarily mean close. In the

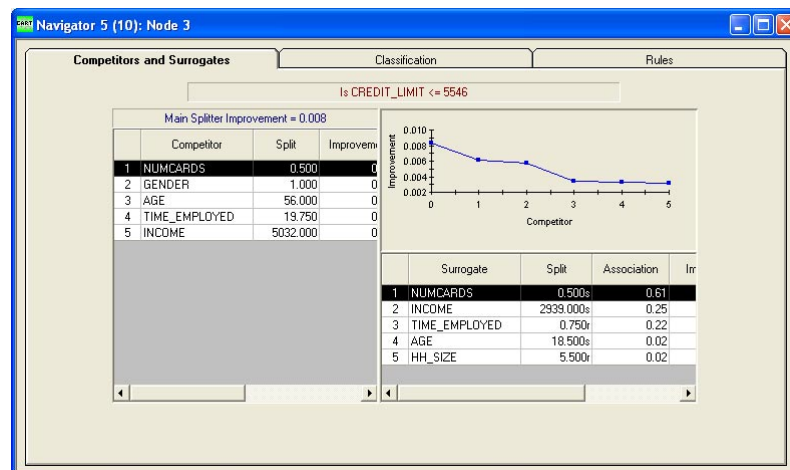
example, the top surrogate has an association score of 0.13 (on a scale of 0.00 to 1.00), which is a rather weak association. (You can think of the association as akin to correlation, but scores above 0.20 represent a good degree of matching.)

When a splitter does not have any close matching surrogates it means that the information content of that variable is unique and cannot be easily substituted for by any other variable. In this example, it should not be surprising to learn that the credit bureau variable N_INQUIRIES contains unique information not reflected in the other variables.

The top five surrogates are ranked by association score and are listed in the bottom-right panel, along with the splitting criterion and the improvement yielded by the surrogate split. In this example, the best surrogate, HH_SIZE, has an association value of 0.13, and a low improvement of 0.0007. The next surrogate, GENDER, is ranked 2nd because of its association score but offers a much better improvement.

✎ Surrogates play the role of splitter when the primary splitter is missing. They play the role of “backup splitter” and are consulted in order. If both the primary and first surrogate splitter are missing, CART would make use of the 2nd ranked surrogate.

More effective surrogates are found in internal node 3 (go left twice from the root and double click).



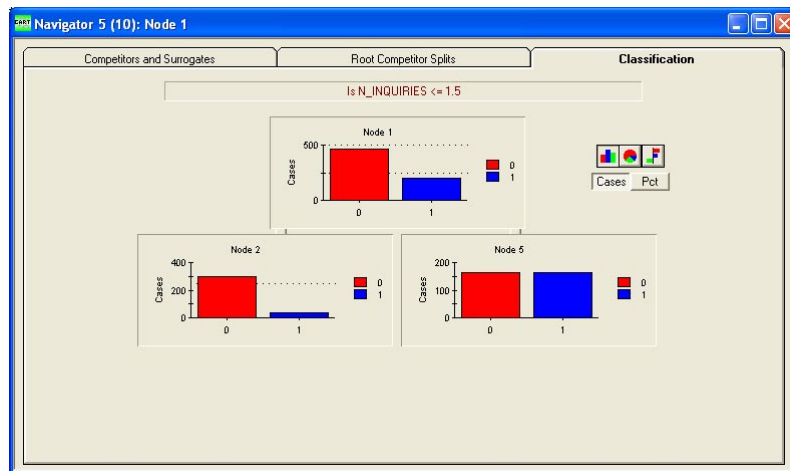
Here the main splitter is CREDIT_LIMIT and the top surrogate NUMCARDS has a strong association score of 0.61. This means that if NUMCARDS were used in place of CREDIT_LIMIT it would partition the data in a similar way and achieve a similar but lower improvement score. In this node the top competitor is also the top surrogate, but you should not expect to see this pattern often.

- See the main reference manual for a detailed discussion of association and improvement.

The Classification tab

The classification tab displays node frequency distributions in a bar graph (or, optionally, a pie chart or horizontal bar chart) for the parent-, left -and right-child nodes. If you use a test sample, frequency distributions for learn and test samples can be viewed separately using the **[Learn]** or **[Test]** buttons.

Below we show the report for the root node. The left child is now clearly dominated by GOODS and the right child contains an equal number of GOODS and BADs.

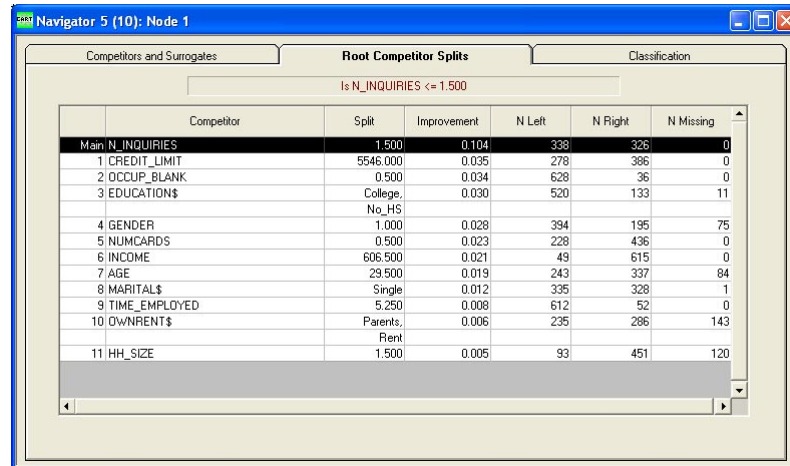


The window offers a choice between bar charts, pie charts and a horizontal bar chart embedding the sample split. You can switch between counts and percentages by pressing the **[Cases]** or **[Pct]** buttons. The horizontal bar chart offers an alternative view of the class partitions. Each colored bar represents one target class. The vertical line shows how the class was partitioned between two children, with the percentage of the class going to the left child shown on the left side and the percentage of the class going to the right child shown on the right side. In this example, less than 20% of Class 1 went to the left side and more than 80% went to the right side.

The Root Competitor Splits tab

In the root node a splitter has access to all the data. Thus, we have a special interest in the performance of variables as splitters in the root. This report lists every variable available for splitting and includes this additional information:

- ♦ N missing: Count of number of records missing data for this variable
- ♦ N left/N right: Count of records going to the left and right children



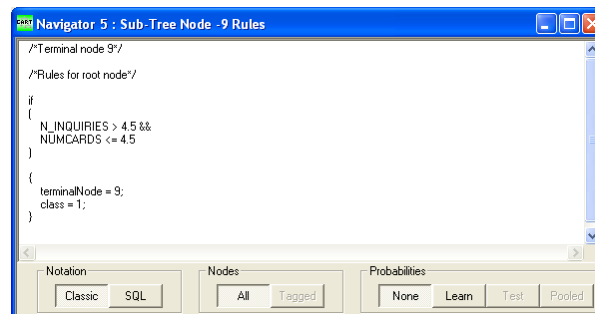
The screenshot shows the 'Root Competitor Splits' tab in the CART Navigator 5 (10): Node 1 window. The table displays the results of a split on the variable 'N_INQUIRIES' with the rule 'Is N_INQUIRIES <= 1.500'. The table has columns for Competitor, Split, Improvement, N Left, N Right, and N Missing.

Competitor	Split	Improvement	N Left	N Right	N Missing
Main N_INQUIRIES	1.500	0.104	338	326	0
1 CREDIT_LIMIT	5546.000	0.035	278	386	0
2 OCCUP_BLANK	0.500	0.034	628	36	0
3 EDUCATION\$	College, No_HS	0.030	520	133	11
4 GENDER	1.000	0.028	394	195	75
5 NUMCARDS	0.500	0.023	228	436	0
6 INCOME	606.500	0.021	49	615	0
7 AGE	29.500	0.019	243	337	84
8 MARITAL\$	Single	0.012	335	328	1
9 TIME_EMPLOYED	5.250	0.008	612	52	0
10 OWNRENT\$	Parents, Rent	0.006	235	286	143
11 HH_SIZE	1.500	0.005	93	451	120

In some circumstances you may be uncomfortable with a main splitter because it is too frequently missing or because it generates a highly uneven split. For example, OCCUP_BLANK puts 628 cases on the left and only 36 cases on the right. OWNRENT\$ has 143 cases missing. Other sections of the manual discuss what you can do if your main splitter exhibits such possibly undesirable characteristics.

The Rules tab

The rules tab will display text rules describing how to reach the node selected, and thus is available for every node except the root. Select Terminal node 9 from the 10-node tree, double click on the node and then select the Rules tab to see:



The screenshot shows the 'Sub-Tree Node -9 Rules' window. It displays the rules for terminal node 9, which are: 'N_INQUIRIES > 4.5' AND 'NUMCARDS <= 4.5'. The window also has tabs for Notation (Classic, SQL), Nodes (All, Tagged), and Probabilities (None, Learn, Test, Pooled).

```

/*Terminal node 9*/
/*Rules for root node*/

if
(
  N_INQUIRIES > 4.5 &&
  NUMCARDS <= 4.5
)
{
  terminalNode = 9;
  class = 1;
}
  
```

Node 9 contains the data segment satisfying the rules:

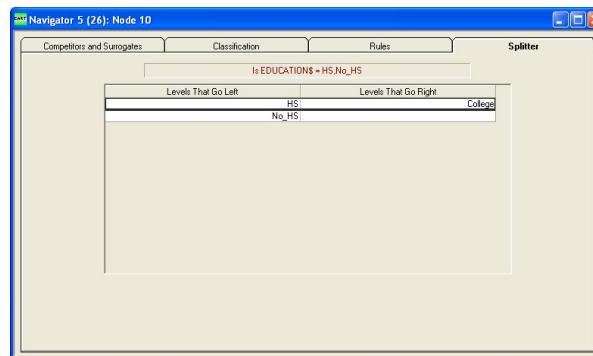
N_INQUIRIES > 4.5 AND
NUMCARDS <= 4.5

and is estimated to be 70% BAD. We need to click on one of the **Probabilities** buttons if we want them to be displayed with the rules.

The rules are formatted as C-compatible code to facilitate applying new data to CART models in other applications. The rule set can be exported as a text file, cut and pasted into another application, and/or sent to the printer. This topic is discussed further below in the section titled "Displaying and Exporting Tree Rules."

The Splitter tab

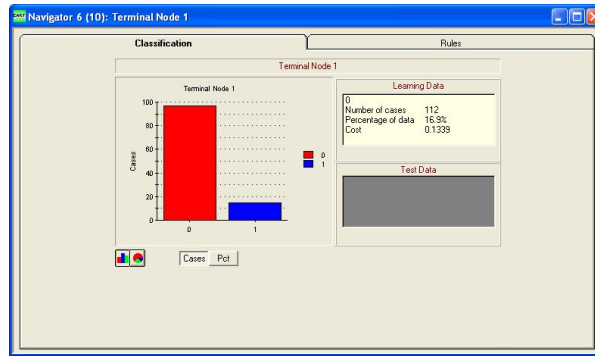
When a node is split on a categorical variable, an additional tab called "Splitter" is available in the Node Information window for all internal nodes. In our example, we will not see a categorical splitter in the tree unless we expand the tree out to 26 nodes. If you do that and go to the parent of Terminal Node 3 (at the bottom left) you will see that it splits on the categorical EDUCATION\$ variable. Click that node and select the **Splitter** tab to obtain:



With only three education levels we can readily see whether a level goes to the left or the right. This report is most useful for following high-level categorical splits or for tracing which levels end up where when the same categorical variable is used as the main splitter multiple times.

Terminal Node Report

To view node-specific information just single click the terminal node of your choice (or right-click and select **Node Report**). A frequency distribution for the classes in the terminal node is displayed as a bar graph (or, optionally, a pie chart), as shown below for the left-most terminal node, Terminal Node 1. Summary node information—class assignment, number of cases in the node, percentage of the data in the node, and misclassification cost—is also displayed for the learn data (and, if you use a test sample, for the test data).



Saving the Navigator/Grove File

To save the Navigator so that you can subsequently reopen the file for further exploration in a later CART session, first make sure that the navigator is your active window (click anywhere on the navigator). Then select **Save Grove...** from the **File->Save...** menu (or press the **[Save Grove...]** button in the Navigator window). In the **Save** dialog window, click on the File Name text box to change the default file name (in this case, the data set name, GOODBAD). The file extension is by default **.grv** and should not be changed. Specify the directory in which the Navigator/Grove file should be saved and then click on **[Save]**.

Previous versions of CART saved two types of tree files: navigator files (with extensions like **.nav** or **.nv3**) and grove files. CART 6.0 stores the navigator inside the grove file and no longer makes use of a separate navigator file format. CART 6.0 will recognize and read old navigator files and you can load these from the **File-Open-Open Navigator** menu selection.

If the trees you are building are large (e.g., several thousand terminal nodes), Windows' system resources can become depleted. To avoid memory problems, consider periodically closing the open Navigator windows you will not need.

More Navigator Controls

- ◆ standard Relative Cost curve
- ◆ color-coded Relative Cost curve
- ◆ percent population by node display

The first two displays show the relative cost curve depending on the number of terminal nodes, while the last display reports how the original data set is distributed into the terminal nodes in the currently-selected tree.

✂ If you click on an individual bar in the “percent population by node” display, the corresponding node in the tree topology is briefly highlighted.

Pressing on the **[Smaller]** or **[Larger]** button causes the scale of the tree topology in the top half of the navigator window to become larger or smaller. This is useful when analyzing large trees.

When applicable, you may switch between learn or test counts displayed for each node by pressing the **[Learn]** or the **[Test]** buttons. Since cross validation was used in this example, only learn counts are available on the node-by-node basis.

You can also save the Navigator or Grove file (needed for scoring) by pressing the **[Save Grove...]** button, or you may translate CART models into SAS®, C, or PMML representations by activating the **[Translate...]** button. Finally, you may apply any tree to data using the Score dialog accessed via the **[Score...]** button. See Chapter 7 for step-by-step instructions for scoring new data.

CART Text Output

The classic text output window contains the detailed technical log that will always be produced by the non-GUI CART running on UNIX, Linux, and mainframe platforms. Most modelers can safely ignore this window because the same information is reported in the GUI displays we have been demonstrating in this tutorial. The classic text output will contain some exclusive reports and advanced information of interest to experienced modelers.

To turn to the text output, select **Classic Output** (shortcut: Ctrl-Alt-C) from the **Window** menu, or click on the window if you can see it. The classic output contains an outline panel on the left with hyperlinks for jumping to the specific locations. Below we selected the first topic in the outline: **Target Frequency Table**.

The screenshot shows a software window titled "Classic Output (Ctrl+Alt+C)". On the left is a "Report Contents" sidebar with a tree structure. The main area displays the "Target Frequency Table" and "Descriptive Statistics" for the variable "TARGET".

Report Contents (Tree 1):

- Target Frequency Table
- Descriptive Statistics
- Missing Value Prevalence
- Tree Sequence
- Node Information
- Terminal Nodes
- Misclassification
- Class Table (CV)
- Class Table (Learn)
- Class Table (Learn)
- Importance
- Option Settings
- Competitor List
- Learn Sample Gains and ROC for TARGET

Target Frequency Table:

```

Variable: TARGET
N Classes: 2
Data Value      N      %      Wgt Count      %
-----
0                461   69.43      461   69.43
1                203   30.57      203   30.57
Total            664
  
```

Descriptive Statistics:

Variable	N	Mean	SD	Min	Max	Sum	Prop Miss
Overall							
AGE	580.00	32.002	7.253	9.000	58.000	18561.000	0.127
CREDIT_LIMIT	664.00	14966.574	20699.596	0.000	104622.000	9937805.000	0.000
GENDER	589.00	-0.689	4.710	-4.000	6.000	-406.000	0.113
HH_SIZE	544.00	3.314	1.567	1.000	7.000	1803.000	0.181
INCOME	664.00	4081.373	2629.637	0.000	20800.000	2710032.000	0.000
N_INQUIRIES	664.00	2.985	3.734	0.000	23.000	1982.000	0.000
NUMCARDS	664.00	1.801	1.839	0.000	9.000	1196.000	0.000
OCCUP_PLANK	664.00	0.054	0.227	0.000	1.000	36.000	0.000
TIME_EMPLOYED	664.00	1.746	3.282	0.500	33.000	1159.500	0.000

You can save a copy of the text output as a record of your analysis by selecting **Save Output...** from the **File->Save** menu. You can also copy and paste sections of the output into another application or to the clipboard. The font used in the Report window can be changed by selecting **Fonts...** from the **Edit** menu. Use a mono-spaced font such as Courier to maintain the alignment of tabular output.

- ✂ You can always regenerate most of the classic output from a saved Grove file by using the **TRANSLATE** facility built into every grove.
- ✂ Advanced users may want to use PERL scripts to process the classic output to create custom reports.

For a line-by-line description of the text output, consult the main reference manual.

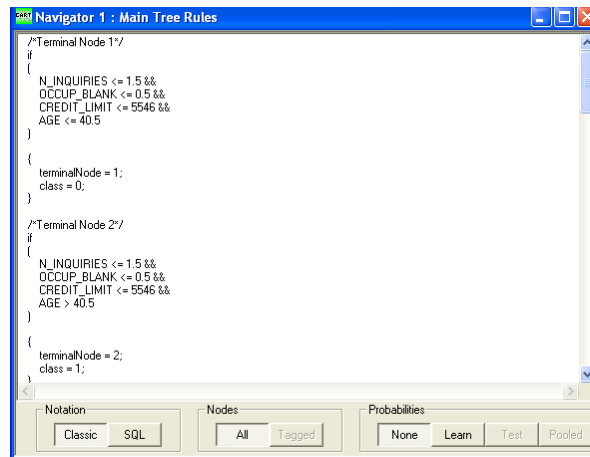
Displaying and Exporting Tree Rules

Decision trees can be viewed as flow charts or as sets of rules that define segments of interest in a database. The rules for a CART tree can be rendered in two quite different ways:

- ◆ As simple rules that are easy to read and understand (approximate model)
- ◆ As complex computer code (more accurate model)

This section focuses on the first form of the rules. The second form is discussed in the sections on scoring and translation.

Every node displayed in a navigator can be described by the rules that lead to it from the root. To view the rules just right click on the node and select **Rules**. If you select the root node for rule extraction you actually get the rules for every terminal node in the tree. Below we show this for our example.



You have a few further options on this window:

- ◆ The rules can be displayed as standard C or SQL programming code.
 - ◆ Probabilities can be based on Learn data, Test data (if available), or on the combination of learn and test data.
 - ◆ Rules can be displayed for specific nodes only (those you have tagged on the navigator via the right mouse click menu).
- 🔍 This rules display is intended only as a rough guide. The rules produced are only an approximate version of the CART model because they do not contain information about surrogate splits. You should use the Translate feature (available by pressing the **[Translate...]** button in the Navigator window) to get the complete representation of the CART model, including surrogates. See Chapter 7 for details.

Scoring Data

There are many reasons to score data with a CART model. You might want to run a quick test of the model's predictive power on new data, or you might actually embed your model into a business process. CART gives you several options for doing this:


- ◆ CART can score data from any source using any previously-built CART model. All you need to do is to attach to your data source, let CART know which grove file to use, and decide where you want the results stored.

- ♦ CART scoring engines are available for deployment on high performance servers that can rapidly process millions of records in batch processes.
- ♦ You can TRANSLATE your model into one of several programming languages including C, SAS, and PMML. (Java may be available by the time you read this.) The code produced needs no further modification and is ready to be run in accordance with the instructions provided in the main reference manual.

To score data using a model you have just built proceed as follows:

1. Press **[Score...]** in the Navigator window containing the model you want to apply.
2. In the **Score Data** window:
 - Accept the current data file or change it using the **[Select...]** button in the **Data** section.
 - Accept the current Grove file (embedded into the current Navigator) or use **[Select...]** to load another one (assuming that it was saved using the **[Save Grove...]** button) in the **Grove** section.
 - Check the **Save results to a file** checkbox and specify the output data set name.
 - Choose the tree you want to apply by pressing the **[Select...]** button in the Subtree section; by default, CART offers the optimal tree.
 - Set the target, weight, and id variables when applicable.
 - Press **[OK]**.
3. The output data set will contain new variables added by CART, including node assignment, class assignment, and predicted probabilities for each case.


New Analysis

To build another tree using the same data set, select **Construct Model...** from the **Model** menu (or click , the "Model Setup" toolbar icon). CART retains the prior model settings in the Model Setup dialogs.

To use another data set, select **Data File...** from the **File->Open** menu. The new selected file will replace the file currently open and all dialog box settings will return to default values.

Saving the Command Log

Although we have used the mouse to make menu selections and to set up and run our models, underneath it all CART is actually generating and executing commands. While you do not ever have to learn how to use these commands, they serve one crucial function for everyone: the commands corresponding to a session are your *audit trail* and permanent record of your actions. If you find that you must reproduce a model or analysis, the command log will ensure that this is possible.

To save the Command Log, select **Open Command Log...** from the **View** menu (or press , the "Command Log" toolbar icon) and then select **Save** from the **File** menu. Specify a directory and the name of the command file, saved by default with a .CMD extension.

The commands can also help accelerate your work. Once you have set up a model with controls that work well for your data, you can use saved (edited) command logs to instantly recreate your working setup. This way you can guarantee that you are including exactly the same list of predictors as you used previously and that you are using your preferred controls.

See Chapter 12 and 13 for more about the CART command log and running CART in batch mode. See also Appendix I for a quick reference to the command line-menu equivalents.

- ✎ CART automatically logs every command associated with your session and automatically saves it to a dedicated file in your CART temporary folder (specified in **Edit->Options->Directories**). This file will be saved even if your computer crashes for any reason, and in the worst case scenario it will be missing only your last command.
- ✎ The name of this file starts with "CART" followed by month and day, followed by hour (military convention 0:23), minutes, and seconds, followed by two underscores. For example, CART1101173521__.TXT refers to the CART session that was finished on November 1st, at 5:35:21 pm. This serves as a complete audit trail of your work with the CART application.
- ✎ The number of session command logs that can be saved to the CART temporary files folder has no limit.

Chapter

4



Classification Trees

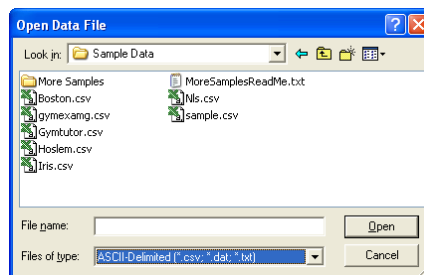
A Biomedical Example

Building Classification Trees

We start by walking through a simple classification problem taken from the biomedical literature. The topic is low birth weight of newborns. The task is to understand the primary factors leading to a baby being born significantly under weight. The topic is considered important by public health researchers because low birth weight babies can impose significant burdens and costs on the healthcare system. A cutoff of 2500 grams is typically used to define a low birth weight baby.

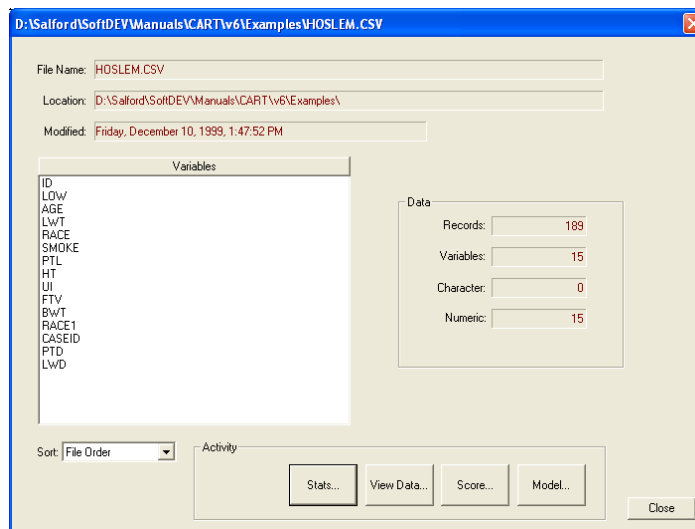
Begin by looking for the HOSLEM.CSV data file that should be located in your Sample Data folder. The CART installer normally creates a Sample Data directory for you under your CART 6.0 directory. If you cannot locate the file you may need to rerun the installer, requesting that it install only the sample data files.


Using the **File-Open>Data File...** menu selections you should see a screen something like the following.



Note the bottom portion of the window that specifies “Files of type:” and the “ASCII-Delimited (*.csv, *.dat, *.txt)” description. If you see a different type of file selected in your window, click the pull down arrow and select the ASCII file type to see the file we need.

Selecting HOSLEM.CSV will bring up the activity screen that provides some basic information about your file, lists the available variables, and allows you to jump to the several other activities.



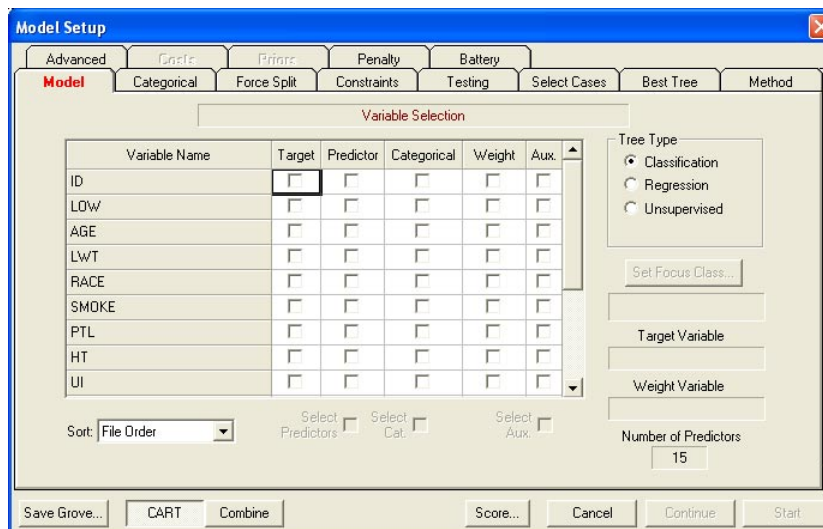
You can always bring up this activity window up by clicking on the  icon on your toolbar.

Definitions of the variables are given below.

LOW	Birth weight less than 2500 grams (coded 1 if <2500, 0 otherwise)
AGE	Mother's age
FTV	Number of first trimester physician visits
HT	History of hypertension (coded 1 if present, 0 otherwise)
LWD	Mother's weight at last menstrual period less than 110 lbs. (coded 1 if <110, 0 otherwise)
PTD	Occurrence of pre-term labor (coded 1 if present, 0 otherwise)
RACE	Mother's ethnicity (coded 1, 2 or 3)
SMOKE	Smoking during pregnancy (coded 1 if smoked, 0 otherwise)
UI	Uterine irritability (coded 1 if present, 0 otherwise)

As you might guess we are going to explore the possibility that characteristics of the mother, including demographics, health status, and the mother's behavior, might influence the probability of a low birth weight baby.

Later we will look into viewing the data and obtaining summary statistics, graphical displays and histograms. Right now let's click the **[Model...]** button that brings up the Model Setup dialog:



The dialog offers 13 tabs that allow you to control all details governing the modeling process. Fortunately, you can set up a model with as few as two mouse clicks. The options are there only for those who need them. Here is a brief description of each tab:

Model	identifies target variable and select predictors
Categorical	notes which numeric predictors are categorical (unordered)
Force Split	dictates which variable should be used to split a node
Constraints	specifies splitter variable criteria
Testing	specifies which test method to use
Select Cases	selects records to use
Best Tree	specifies best tree selection method
Method	specifies splitting rule to use to grow tree
Cost	specifies cost of making specific mistakes
Priors	specifies how to balance unequal classes
Penalty	sets penalties on predictors, missing values, categoricals
Advanced	specifies other model-building options
Battery	specifies modeling automation

The only **required** step is to specify a target variable and tree type in the **Model Setup—Model** tab.

For most users the default settings for any tab are reasonable and suffice to obtain useful models with good to excellent performance. As you become more accustomed to the software you might experiment with the available controls to see if you can improve your results. We also provide automatic experimentation for you using the **Battery** tab, described in detail later.

If the other Model Setup dialog tabs are left unchanged, the defaults used are:

- ◆ All variables in the data set other than the target will be used as predictors (the **Model** tab)
- ◆ No weights will be applied (the **Model** tab)
- ◆ 10-fold cross validation for testing (the **Testing** tab)
- ◆ Minimum cost tree will become the best tree (the **Best Tree** tab)
- ◆ Only five surrogates will be tracked and they will all count equally in the variable importance formula (the **Best Tree** tab)
- ◆ GINI splitting criterion for classification trees and least squares for regression trees (the **Method** tab)
- ◆ Unit (equal) misclassification costs (the **Costs** tab)
- ◆ Equal priors: all classes treated as if they were equal size (the **Priors** tab)
- ◆ No penalties (the **Penalty** tab)
- ◆ Parent node requirements set to 10 and child node requirements set to 1 (the **Advanced** tab)
- ◆ Allowed sample size set to the currently-open data set size (the **Advanced** tab)

Many other options are available to the advanced user and we invite you to explore them at your leisure in the chapters that follow. The good news about CART is that you can get started by focusing only on the essentials, deferring advanced topics. The remainder of this section discusses the model setup process. Subsequent sections cover additional options.

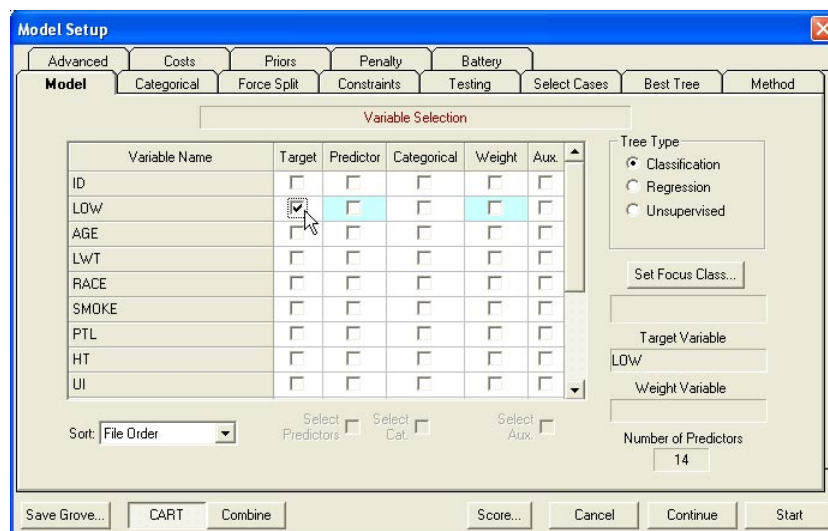
The Model tab

The **Model Setup—Model** tab is the central location for model control—where you identify the target or dependent variables. This is the one and only task that CART requires of you. CART will not know which column of your data to try to analyze without your guidance. Once you provide that information CART is technically able to do everything else for you.

In practice you will probably also want to select the candidate predictor (independent) variables, because data sets typically contain bookkeeping columns such as ID variables that are not suitable for prediction. In some cases you may also have a weight variable. Where possible CART will automatically realize that you want to grow a classification tree. But when the target variable is numeric you do have the choice of growing a classification or regression tree and you may need to correct the selection indicated on the Model Setup dialog. This is the heart of the Model Setup dialog.

Target Variable Selection

The target variable is specified by checking off ONE variable in the target column of the **Model Setup—Model** tab. Locate the row with LOW as **Variable Name** and put a checkmark in the **Target** column.



After the target has been checked, the **Model** tab switches from red to black, indicating that CART is ready to start an analysis according to the default settings.

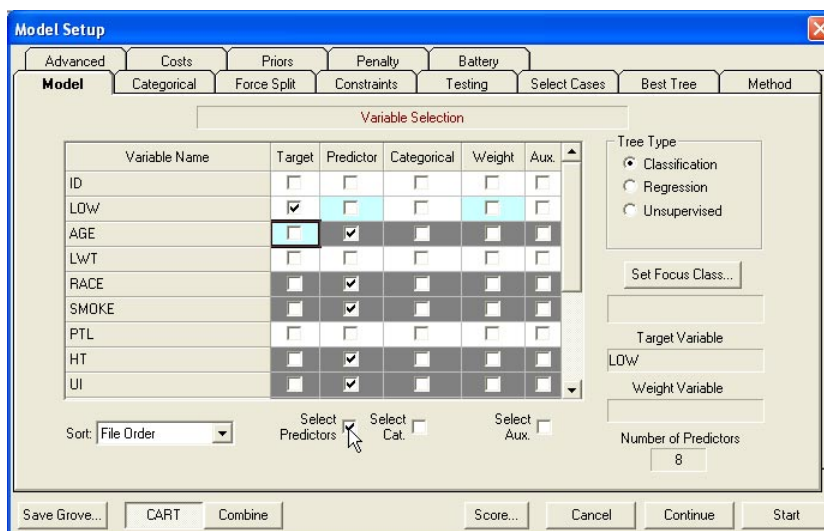
Specifying Tree Type

CART uses a set of **Tree Type** radio buttons to determine if the tree grown will be a **classification** tree or a **regression** tree. The difference between the two tree types is simple. Classification trees use a "categorical" target variable (e.g., YES/NO, while the regression tree uses a "continuous" target variable (such as AGE or INCOME). The purpose of classification is to accurately discriminate between (usually a small number of) classes; the purpose of regression is to predict values that are close to a true outcome (with usually a large number or even an infinity of possible outcomes).

When the **Tree Type: Classification** radio button is checked, the target variable automatically will be considered categorical regardless of the Categorical check-box designation defined in Model tab. Similarly, the **Regression** radio button will automatically cancel the categorical status of the target variable (so long as the variable is coded as a number and not as text). In other words, the specified **Tree Type** determines whether a numeric target is treated as categorical or continuous, superseding any Categorical check-box designation.

Predictor Variable Selection

Candidate predictor (independent) variables are specified by check marks in the Predictor column. In this example, include the following subset of variables as predictors: AGE, RACE, SMOKE, HT, UI, FTV, PTD, and LWD, by placing checkmarks in the **Predictor** column against the above variables. Alternatively, hold down the **<Ctrl>** key to simultaneously highlight the variables with left-mouse clicks and then place a checkmark in the **Select Predictors** box at the bottom of the column. The **Model** tab will appear as follows:



If you inadvertently include a variable as a predictor, simply uncheck the corresponding box.

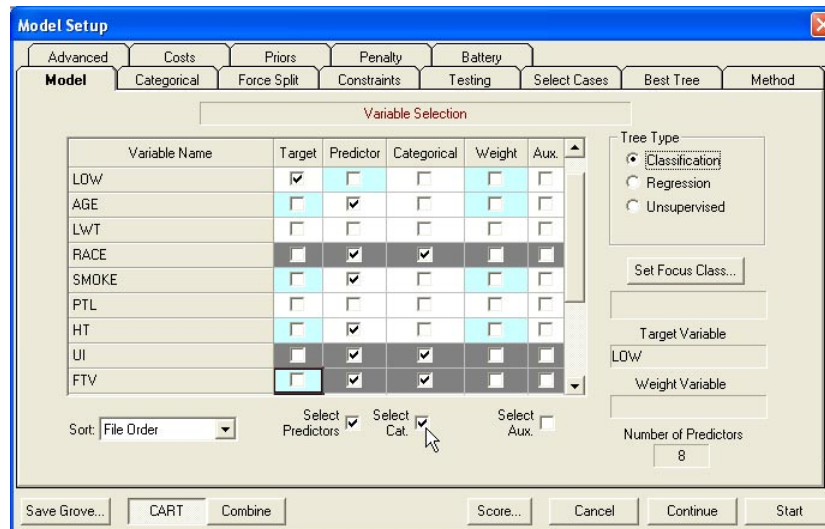
✂ Note also that each of the model setup tabs contains a **[Save Grove...]** button in the lower left corner. This allows you to request saving the model for future review, scoring, or export.

✂ For command-line users, the **MODEL** command sets the target variable, while the **KEEP** command defines the predictor list. See the following command line syntax.

```
MODEL <depvar>
KEEP < indep_var1, indep_var2, ...,indep_var#>
-----
MODEL LOW
KEEP AGE, RACE, SMOKE, HT, UI, FTV, PTD, LWD
```

Categorical Predictors

Put checkmarks in the Categorical column against those predictors that should be treated as categorical. For our example, specify RACE, UI, and FTV as categorical predictor variables. Alternatively, as for predictor variables, hold down the **<Ctrl>** key to simultaneously highlight the variables with left-mouse clicks and then place a checkmark in the **Select Categorical** box at the bottom of the column.



When the **Tree Type: Classification** radio button is checked, the target variable will be automatically defined as categorical and appear with the corresponding checkmark at later invocations of the Model Setup. Similarly, the **Regression** radio button will automatically cancel the categorical status of the target variable. In other words, the specified **Tree Type** determines whether the target is treated as categorical or continuous.

Annotation On Categorical Variables

Categorical targets and predictors are those that take on a conceptually finite set of discrete values, for example, data naturally in text form (e.g., "Male," "Female"). You may declare any variable categorical but you should do so only when this is sensible.

It should be noted that CART 6 supports "high-level categoricals" through its proprietary algorithms that quickly determine effective splits in spite of the daunting combinatorics of many-valued predictors. This feature was introduced in CART 4 and is increasingly important considering CART 6's character predictors, which in "real world" datasets often have hundreds or even thousands of levels. When forming a categorical splitter, traditional CART searches all possible combinations of levels, an approach in which time increases geometrically with the number of levels. In contrast, CART's high-level categorical algorithm increases linearly with time, yet

yields the optimal split in most situations. See the section below titled "High-Level Categorical Predictors" for additional details.

Character Variable Caveats

Character variables are implicitly treated as categorical (discrete), so there is no need to "declare" them categorical. CART 6 has no internal limit on the length of character data values (strings). You are limited in this respect only by the data format you choose (e.g., SAS, text, Excel, etc.).

- ✗ Character variables (marked by "\$" at the end of variable name) will always be treated as categorical and cannot be unchecked.
- ✗ Occasionally columns stored in an Excel spreadsheet will be tagged as "Character" even though the values in the column are intended to be numeric. If this occurs with your data refer to the READING DATA section to remedy this problem.

Categorical vs. Continuous Predictors

Depending whether a variable is declared as continuous or categorical, CART will search for different types of splits. Each takes on a unique form.

Continuous Split Form

Continuous splits will always use the following form.

A case goes left if
 $[split-variable] \leq [split-value]$




A node is partitioned into two children such that the left child receives all the cases with the lower values of the $[split-variable]$.

Categorical Split Form

Categorical splits will always use the following form.

A case goes left if
 $[split-variable] = [level_i \text{ OR } \dots level_j \text{ OR } \dots level_k]$

- ✗ In other words, we simply list the values of the splitter that go left (and all other values go right).
- ✗ If a categorical variable with many levels is coded as a number it may actually be helpful to treat it as a continuous variable. This is discussed further in a later chapter.

-  One should exercise caution when declaring continuous variables as categorical because a large number of distinct levels may result in significant increases in running times and memory consumption.
-  Any categorical predictor with a large number of levels can create problems for the model. While there is no hard and fast rule, once a categorical predictor exceeds about 50 levels there are likely to be compelling reasons to try to combine levels until it meets this limit. We show how CART can conveniently do this for you later in the manual.
-  For command-line users, categorical variables are defined using the **CATEGORY** command. See the following command line syntax.





```
CATEGORY <cat_var1, cat_var2, ..., cat_var#>
-----
CATEGORY LOW, RACE, SMOKE, UI
```

Case Weights

In addition to selecting target and predictor variables, the **Model** tab allows you to specify a case-weighting variable.

Case weights, which are stored in a variable on the dataset, typically vary from observation to observation. An observation's case weight can, in some sense, be thought of as a repetition factor. A missing, negative or zero case weight causes the observation to be deleted, just as if the target variable were missing. Case weights may take on fractional values (e.g., 1.5, 27.75, 0.529, 13.001) or whole numbers (e.g., 1, 2, 10, 100).

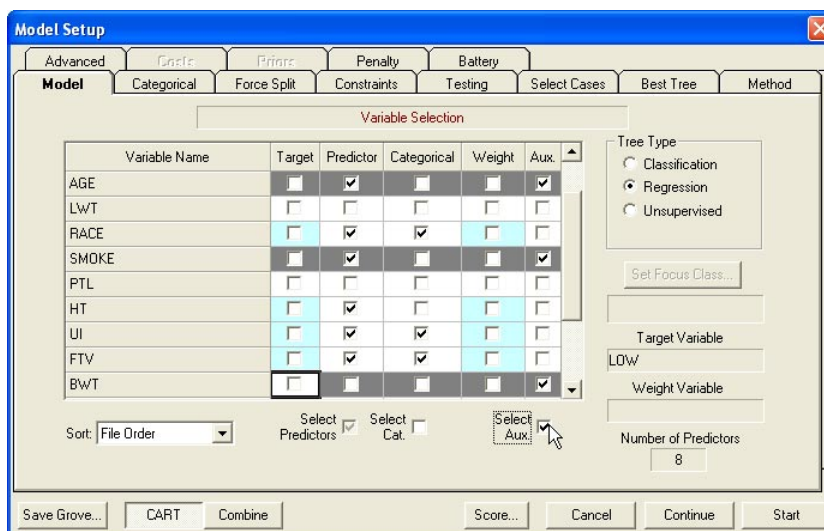
To select a variable as the case weight, simply put a checkmark against that variable in the **Weight** column.

-  Case weights do not affect linear combinations in CART-SE, but are otherwise used throughout CART. CART-Pro and ProEX include a new linear combination facility that recognizes case weights.
-  If you are using a test sample contained in a separate dataset, the case weight variable must exist and have the same name in that dataset as in your main (learn sample) dataset.
-  For command line users, the variable containing observation case weights is specified with the **WEIGHT** command, which is issued after the **USE** command and before the **BUILD** command. See the following command line syntax:


```
WEIGHT <wgtvar>
```


Auxiliary Variables

Auxiliary variables are variables that are tracked throughout the CART tree but are not necessarily used as predictors. By marking a variable as Auxiliary you indicate that you want to be able to retrieve basic summary statistics for such variables in any node in the CART tree. In our modeling run based on the HOSLEM.CSV data, we mark AGE, SMOKE and BWT as auxiliary.



Later in this chapter, in the section titled "Viewing Auxiliary Variable Information," we discuss how to view auxiliary variable distributions on a node-by-node basis.



Command-line users will use the following command syntax to specify auxiliary variables.

```
AUXILIARY <auxvar1>, <auxvar2>, ... etc.
-----
AUXILIARY AGE, SMOKE, BWT
```

Setting Focus Class

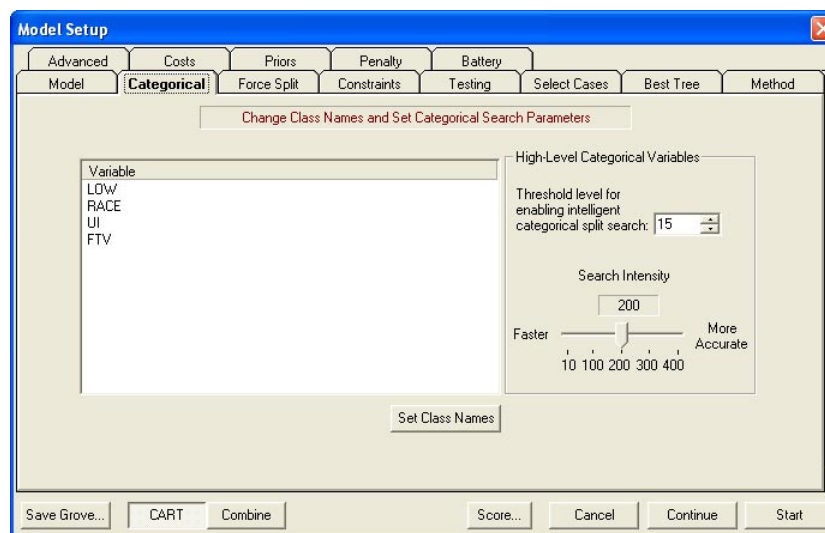
In classification runs some of the reports generated by CART (gains, prediction success, color-coding, etc.) have one target class in focus. By default, CART will put the first class it finds in the dataset *in focus*. A user can overwrite this by pressing the **[Set Focus Class...]** button.

Sorting Variable List

The variable list can be sorted either in physical order or alphabetically by changing the **Sort:** control box. Depending on the dataset, one of those modes will be preferable, which is usually helpful when dealing with large variable lists.

The Categorical tab

The Categorical tab allows you to manage text labels for categorical predictors and it also offers controls related to how we search for splitters on high-level categorical predictors. The splitter controls are discussed later as this is a rather technical topic and the defaults work well.



Setting Class Names

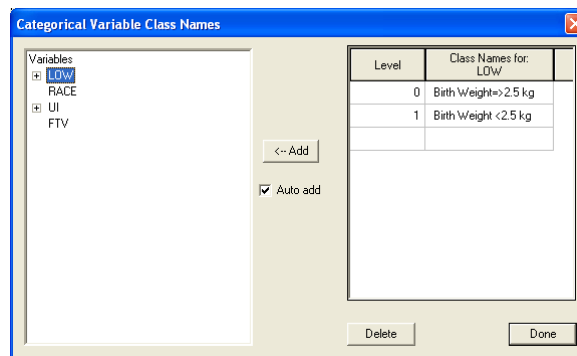
Class names are defined in the Categorical tab. Press **[Set Class Names]** to get started. In the left panel, select a variable for which labels are to be defined. If any class labels are currently defined for this variable, they will appear in the left panel and, if the variable is selected, in the right panel as well (where they may be altered or deleted). To enter a new class name in the right panel for the selected variable, define a numeric value (one that will appear in your data) in the **"Level"** column and its corresponding text label in the **"Class names for:"** column. Repeat for as many class names as necessary for the selected variable.

You need not define labels for all levels of a categorical variable. A numeric level, which does not have a class name, will appear in the CART output as it always has, as a number. Also, it is acceptable to define labels for levels that do not occur in your

data. This allows you to define a broad range of class names for a variable, all of which will be stored in a command script (.CMD file) , but only those actually appearing in the data you are using will be used.

In a classification tree, class names have the greatest use for categorical numeric target variables (i.e., in a classification tree). For example, for a four-level target variable PARTY, classes such as "Independent," "Liberal," "Conservative," and "Green" could appear in CART reports and the navigator rather than levels "1", "2", "3", and "4." In general, only the first 32 characters of a class name are used, and some text reports use fewer due to space limitations.

In our example we specify the following class names for the target variable LOW and predictor UI. These labels then will appear in the tree diagrams, the CART text output, and most displays. The setup dialog appears as follows.



GUI CART users who use class names extensively should consider defining them with commands in a command file and submitting the command file from the CART notepad once the dataset has been opened. The CLASS commands must be given before the model is built.

✂ If you use the GUI to define class names and wish to reuse the class names in a future session, save the command log before exiting CART. Cut and paste the CLASS commands appearing in the command log into a new command file.

📄 Command-line users will use the following command syntax to define class names

```
CLASS <variable> <value1> = "<label1>",
      <value2> = "<label2>"...etc.
-----
CLASS LOW 0="Birth Weight=>2.5 kg",
          1="Birth Weight <2.5 kg"
CLASS UI  0 = "Uterine irritability = NO",
          1 = "Uterine irritability = Yes"
```

You can add labels to the target variable AFTER a tree is grown, but these will appear only in the navigator window (not in the text reports). Activate a navigator window, pull down the **View** menu and select the **Assign Class Names...** menu item.

High-Level Categorical Predictors

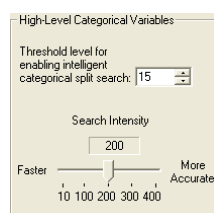
We take great pride in noting that CART is capable of handling categorical predictors with thousands of levels (given sufficient RAM workspace). However, using such predictors in their raw form is generally not a good idea. Rather, it is usually advisable to reduce the number of levels by grouping or aggregating levels, as this will likely yield more reliable predictive models. It is also advisable to impose the HLC penalty on such variables (from the **Model Setup—Penalty** tab). These topics are discussed at greater length later in the manual. In this section we discuss the simple mechanics for handling any HLC predictors you have decided to use.

For the binary target, high-level categorical predictors pose no special computational problem as exact short cut solutions are available and the processing time is minimal no matter how many levels there are.

For the multi-class target variable (more than two classes), we know of no similar exact short cut methods, although research has led to substantial acceleration. HLCs present a computational challenge because of the sheer number of possible ways to split the data in a node. The number of distinct splits that can be generated using a categorical predictor with K levels is $2^{K-1} - 1$. If K=4, for example, the number of candidate splits is 7; if K=11, the total is 1,023; if K=21, the number is over one million; and if K=35, the number of splits is more than 34 billion! Naïve processing of such problems could take days, weeks, months, or even years to complete!

To deal more efficiently with high-level categorical (HLC) predictors, CART has an intelligent search procedure that efficiently approximates the exhaustive split search procedure normally used. The HLC procedure can radically reduce the number of splits actually tested and still find a near optimal split for a high-level categorical.

The control option for high-level categorical predictors appears in the **Model Setup—Categorical** tab as follows.



The settings above indicate that for categorical predictors with 15 or fewer levels we search all possible splits and are guaranteed to find the overall best partition. For predictors with more than 15 levels we use intelligent shortcuts that will find very good partitions but may not find the absolute overall best. The threshold level of 15 for enabling the short-cut intelligent categorical split searches can be increased or decreased in the Categorical dialog. In the short cut method we conduct “local” searches that are fast but explore only a limited range of possible splits. The default setting for the number of local splits to search is around 200. To change this default and thus search more or less intensively, increase or decrease the search intensity gauge. Our experiments suggest that 200 is a good number to use and that little can be gained by pushing this above 400. As indicated in the Categorical dialog, a higher number leads to more intensive and longer searching whereas a lower number leads to faster, less thorough searching. If you insist on more aggressive searching you should go to the command line.



Command-line users will use the following command syntax to define the high-level categorical thresholds.

```
>BOPTIONS NCLASSES = 20  
>BOPTIONS HLC = 600, 10
```

BOPTIONS NCLASSES = 20 turns on shortcut searching for categoricals with more than 20 levels

BOPTIONS HLC = 600, 10 conducts 600 local searches, each of which is subjected to a further 10 refinement searches. The default settings of BOPTIONS HLC = 200, 10 should suffice for most problems.

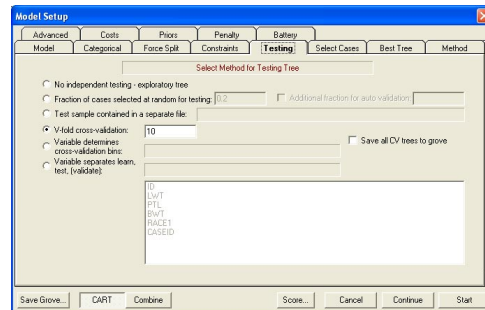
Remember that these controls are only relevant if your target variable has more than two levels. For the two-level binary target (the YES/NO problem), CART has special shortcuts that always work.

Remember that there are actually disadvantages to searching too aggressively for the best HLC splitter, as such searches increase the likelihood of overfitting the model to the training data.

The Testing Tab

Testing is a vital stage in the CART tree selection process, and without testing we cannot know how well a given tree can be expected to perform on new data. CART allows you to choose from five different test strategies accessed in the **Model Setup—Testing** tab, where you will see the following methods:

1. No independent testing
2. V-fold cross validation (default is 10-fold)
3. Fraction of cases to be set aside at random:
for testing (default = 0.20)
for validation (default = 0.00)
4. Test sample contained in a separate file
5. Variable separates learn and test samples (binary indicator)



Default test setting: 10-fold cross validation.

No Independent Testing

This option skips the entire testing phase and simply reports the largest tree grown. We recommend you use this option only in the earliest stages of becoming familiar with the data set, as this option provides no way to assess the performance of the tree when applied to new data. Because no test method is specified, CART does not select an “optimal” tree.

Bypassing the test phase can be useful when you are using CART to generate a quick cross tabulation of the target against one of your predictors. It is also useful for “supervised binning” or aggregation of variables such as high-level categoricals. This use of CART is discussed in more detail in other sections.

V-fold Cross validation

Cross validation is a marvelous way to make the maximal use of your training data, although it is typically used when data sets are small. For example, because the HOSLEM data set contains only 189 records, it would be painful to segregate some of those data for the sake of testing alone. Cross validation allows you to build your tree using all the data. The testing phase requires running an additional 10 trees (in 10-fold CV), each of which is tested on a different 10% of the data. The results from those 10 test runs are combined to create a table of synthesized test results.

Cross validation is discussed in greater detail in the command line manual and in the references cited there. When deciding whether or not to use cross validation, keep these points in mind:

- ✂ Cross validation is always a reasonable approach to testing. However, it is primarily a procedure that substitutes repeated analyses of different segments of your data for a more typical train-test methodology. If you have plentiful data you can save quite a bit of time by reserving some of the data for testing.

Cross validation can give you useful reports regarding the sensitivity of results to small changes in the data.

Even in a large data set the class of interest may have only a handful of records. When you have only a small number of records in an important target class you should think of your data set as small no matter how many records you have for other classes. In such circumstances, cross validation may be the only viable testing method.

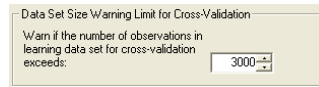
Reducing the number of cross validation folds below ten is generally not recommended. In the original CART monograph, Breiman, Friedman, Olshen and Stone report that the CV results become less reliable as the number of folds is reduced below 10. Further, for classification problems there is very little benefit from going up to 20 folds.

If there are few cases in the class of interest you may need to run with fewer than 10 CV folds. For example, if there are only 32 YES records in a YES/NO classification data set (and many more NOs) then eight-fold cross validation would allow each fold to contain four of these cases. Choosing 10-fold for such data would probably induce CART to create nine folds with three YES records and one fold with five YES records. In general, the better balance obtained from the eight-fold CV would be preferable.

There is nothing technically wrong with two-fold cross validation but the estimates of the predictive performance of the model tend to be too pessimistic. With 10-fold cross validation you get more accurate assessments of the model's predictive power.

- 💡 Every target class must have at least as many records as the number of folds in the cross validation. Otherwise, the process breaks down, an error message is reported, and a “No Tree Built” situation occurs. This means that if your data set contains only nine YES records in a YES/NO problem, you cannot run more than nine-fold cross validation. Modelers usually run into this problem when dealing with, say, a three-class target where two of the classes have many records and one class is very small. In such situations, consider either eliminating rare class cases from the dataset or merging them into a larger class.

If your data set has more than 3,000 records and you select cross validation as your testing method, a dialog will automatically open informing you that you must increase the setting for the “maximum number of observations in learning data set with cross validation” in the **Model Setup—Advanced** tab. This warning is intended to prevent you from inadvertently using cross validation on larger data sets and thus growing eleven trees instead of just one. To raise the threshold, adjust the value in the dialog below:



The advent of the Pentium IV class of CPUs has made run times so short that you can now comfortably run cross validation on much larger data sets.

Fraction of Cases Selected at Random for Testing

Use this option to let CART automatically separate a specified percentage of data for test purposes. Because no optimal fraction is best for all situations, you will want to experiment. In the original CART monograph the authors suggested a 2/3, 1/3 train/test split, which would have you set the test fraction to .33. In later work, Jerome Friedman suggested using a value of .20. In our work with large datasets we favor a value of .50 and in some cases we even use .70 when we want to quickly extract a modest-sized training sample. So our advice is: don't be reluctant to try different values.

 In the command language this value is set with a statement like

```
ERROR P=.20
```

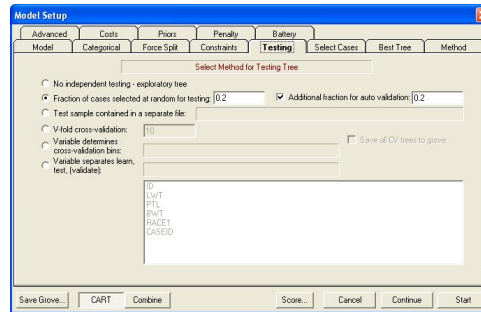
The advantage of using ERROR P=.50 is that the train and test samples are almost identical in size, facilitating certain performance comparisons in individual nodes.

Setting ERROR P=.80, for example, is a fast way to pull a relatively small extract from a large database. Just be sure to check the size of the sample that is selected for training. If it is too small you cannot expect reliable results.

This mechanism does not provide you with a way of tagging the records used for testing. If you need to know which records were set aside for testing you should create a flag marking them for test and then use the SEPVAR method for testing (see below).

Three-way Random Train/Test/Validate Partitions

To request a random division of your data into a three-way partition, just check the relevant box in the **Model Setup—Testing** tab and specify your preferred fractions. When setting up such partitions be sure that each partition will be large enough to fulfill its function. In the example below we have set up a partition that is 60% train, 20% test, and 20% validate.



Test Sample Contained in a Separate File

Two separate files are assumed—one for learning and one for testing. The files can be in different database formats and their columns do not need to be in the same order.

- ✎ The train and test files must both contain ALL variables to be used in the modeling process.
- ✎ In general we recommend that you keep your train and test data in the same file for data management purposes. This helps to ensure that if you process your training data you also process the test data in exactly the same way.

Variable Separates Test (and Validate) Samples


A variable on your data set can be used to flag which records are to be used for learning (training) and which are to be used for testing or validation.


- Use a binary (0/1) numeric variable to define simple learn/test partitions. We like to code such variables with 0 indicating “train” and 1 indicating “test.”
- If you prefer you can use a text variable with the value “TEST” for selected records. The other records can be marked as “TRAIN” or “LEARN.” (You can use lower case if you prefer.)

This option gives you complete control over train/test partitions because you can dictate which records are assigned to which partition during the data preparation process.

For a three-way partition of the data, create a variable with values for “train” “test” and “valid” and select that variable on the testing tab after clicking on the “Variable separates” test method option. In scripts you can use the command like

```
ERROR SEPVAR = TEST_FLAG$
```

 Consider creating several separation variables to explore the sensitivity of the model-building process to random data partition variation.

 Command-line users implement these strategies using one of the following commands:

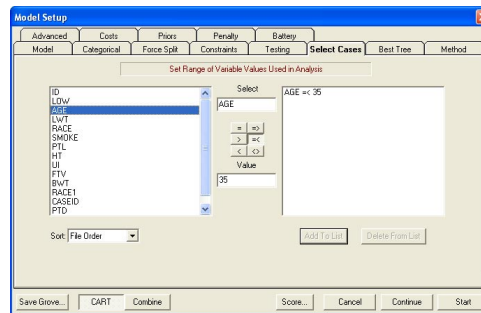
```
ERROR EXPLORATORY
ERROR CROSS=<N>
ERROR PROP=<p>
ERROR FILE=<file_name>
ERROR SEPVAR=<variable>
```

The Select Cases tab

The **Model Setup—Select Cases** tab allows you to specify up to ten selection criteria for building a tree based on a subset of cases. A selection criterion can be specified in terms of any variable appearing in the data set, whether or not that variable is involved in the model, and is constructed as follows:

1. Double-click a variable in the variable list to add that variable to the **Select** text box.
2. Select one of the predefined logical relations by clicking its radio button.
3. Enter a numerical value in the **Value** text box.
4. Click **[Add to List]** to add the constructed criterion to the right window (and use **[Delete from List]** to remove).

For example, if you want to exclude all mothers over 35 years of age from the analysis, double-click on AGE. Click on the **[=<]** button and enter 35 in the **Value** text box. When you click on **[Add to List]**, AGE=<35 will now appear in the previously-blank panel on the right, as illustrated above.



The SELECT criteria are “ANDed,” meaning that if you specify two conditions, both must be satisfied for a record to be selected into the analysis. If you want to create logical selection criteria that allow some but not all conditions to be met you will need to use the built-in BASIC programming language.

Command-line users need to use the following command syntax to specify selection criteria, where *<condition>* is written as *<variable> <relation> <# | 'string>*.

```
SELECT <condition1>, <condition2>, ... etc.
-----
SELECT AGE =< 35
```

Using CART's Built-in Programming Language

As an alternative to the **Model Setup—Select Cases** tab, CART offers a full built-in BASIC programming language. When accessed via the command line, BASIC can be used to modify existing variables as well as to define new variables, filter cases and implement other database programming functions at any step during the Model Setup process. For example, if you are in the Model Setup dialog and want to create a new variable to add to your candidate predictor list, click the **[Continue]** button. Ensure that the command prompt is “on” by placing a checkmark by the **Command Prompt** from the **File** menu item. The command prompt is represented by the “>” character.

At the >, type:

```
%IF FTV>0 THEN LET NEWVAR=1
%ELSE LET NEWVAR=0
```

to create a categorical variable, NEWVAR, that takes on the value 1 if the number of first trimester visits was greater than zero and a value of 0 otherwise. To then add NEWVAR as a candidate predictor variable, reopen the Model Setup dialog. NEWVAR will now appear in the Variables box of the Model dialog; highlight NEWVAR and add it to the predictor list.

✎ The “%” signs are part of the input and signal the command parser that the rest of the line should be treated as a BASIC statement, not as a CART command.

Alternatively, you can use BASIC to take the log or square root (as well as many other mathematical and statistical functions) of an existing variable. BASIC can also be used to draw a random sub-sample from the input data set. By using the uniform random number (URN) generator in BASIC, deleting a random sample of 50 percent, for example, is easily accomplished with the following statement:

```
% IF URN>.5 THEN DELETE
```

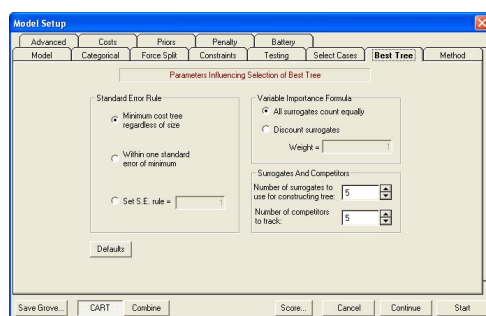
For more about CART’s built-in BASIC programming language, see Appendix IV in the main reference manual.

The Best Tree tab

The **Model Setup—Best Tree** tab is largely of historical interest as it dates to a time when CART would produce a single tree in any run. Specifying how you wanted that single tree to be selected was an important part of the model setup procedure. In today’s CART you have full access to every tree in the pruned tree sequence and you can readily select trees of a size different than considered optimal. Nonetheless, when a tree is saved to a grove, CART always marks one of the pruned sub-trees as optimal. This tree will be selected by default for scoring. When you are working with many trees in a batch-scoring mode it will be most convenient if they are all marked with your preferred method for optimal tree selection.

The Best Tree tab allows you to specify and modify the following parameters influencing the selection of the “best,” or “optimal,” tree:

Default Best Tree settings: minimum cost tree regardless of size, all surrogates count equally, five surrogates used to construct tree.



Standard Error Rule

The standard error rule, the parameter CART uses to select the optimal tree following testing, is specified in the **Best Tree** tab. The default setting is the minimum cost tree regardless of size, that is, the tree that is most accurate given the specified testing method. In certain situations, you may wish to trade a more accurate tree for a smaller tree by selecting the smallest tree within one standard error of the minimum cost tree or by setting the standard error parameter equal to any nonnegative value.

The primary use of the standard error rule is for processing many models in batch mode, or when you do not expect to be able to inspect each model individually. In such circumstances you will want to give some thought to specifying how the best model should be selected automatically. If you are examining each model visually on screen, then the best tree definition is not that important as you can readily select another tree interactively on screen.

Variable Importance Formula

In the Best Tree dialog, you can also specify how variable importance scores are calculated and how many surrogates are used to construct the tree. Rather than counting all surrogates equally, the default calculation, you can fine-tune the variable importance calculation by specifying a weight to be used to discount the surrogates. Click on the **Discount surrogates** radio button and enter a value between 0 and 1 in the **Weight** text box.

Number of Surrogates

After CART has found the best splitter (primary splitter) for any node it proceeds to look for surrogate splitters: splitters that are similar to the primary splitter and can be used when the primary split variable is missing. You have control over the number of surrogates CART will search for; the default value is five. When there are many predictors with similar missing value patterns you might want to increase the default value.

You can increase or decrease the number of surrogates that CART searches for and saves by entering a value in the **Number of Surrogates Used to Construct Tree** box or by clicking on the up/down arrow key.

✎ The number of surrogates that can be found will depend on the specific circumstances of each node. In some cases there are no surrogates at all. Your N surrogates sets limits on how many will be searched for but does not guarantee that this is the number that will actually be found.

If all surrogates at a given node are missing or no surrogates were found for that particular node, a case that has a missing value for the primary splitter will

be moved to the left or right child node according to a default rule discussed later.

Because the number of surrogates you request *can* affect the details of the tree grown we have placed this control on the Best Tree tab. Usually the impact of this setting on a tree will be small, and it will only affect trees grown on data with missing values.



Command-line users will use the following command syntax to set the standard error rule:

```
BOPTIONS SERULE=<value>
```

To discount surrogates, use:

```
BOPTIONS IMPORTANCE=<weight> (weight must be between 0 and 1).
```

To limit the number of surrogates to be kept, use:

```
BOPTIONS SURROGATES=<N>
```

The Method Tab

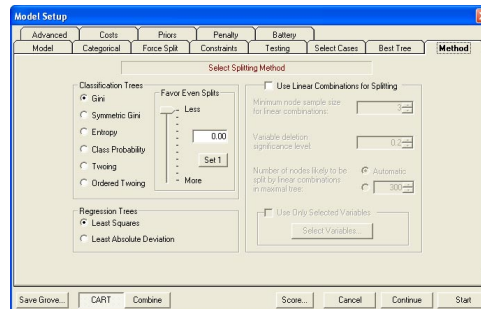
The **Model Setup—Method** tab allows you to specify the splitting rule used to construct the classification or regression tree and to turn on the linear combinations option.

Splitting Rules

A splitting rule is a method and strategy for growing a tree. A good splitting rule is one that yields accurate trees! Since we often do not know which rule is best for a specific problem it is good practice to experiment. For classification trees the default rule is the Gini. This rule was introduced in the CART monograph and was selected as the default because it generally works quite well. We have to agree with the original CART authors: working with many hundreds of data sets in widely different subject matters we have still seen the Gini rule to be an excellent choice. Further, there is often only a small difference in performance among the rules.

However, there will be circumstances in which the performance between, say, the Gini and Entropy is quite substantial, and we have worked on problems where using the Twoing rule has been the only way to obtain satisfactory results. Accuracy is not the only consideration people weigh when deciding on which model to use. Simplicity and comprehensibility can also be important. While the Gini might give you the most accurate tree, the Twoing rule might tell a more persuasive story or yield a smaller

although slightly less accurate tree. Our advice is to not be shy about trying out the different rules and settings available on the **Method** tab.



Here are some brief remarks on different splitting rules:

Gini:

This default rule often works well across a broad range of problems. Gini has a tendency to generate trees that include some rather small nodes highly concentrated with the class of interest. If you prefer more balanced trees you may prefer the results of the Twoing rule.

Symmetric Gini:

This is a special variant of the Gini rule designed specifically to work with a cost matrix. If you are not specifying different costs for different classification errors, the Gini and the Symmetric Gini are identical. See the discussions on cost matrices for more information.

Entropy:

The Entropy rule is one of the oldest decision tree splitting rules and has been very popular among computer scientists. Although it was the rule first used by CART authors Breiman, Friedman, Olshen, and Stone, they devote a section in the CART monograph to explaining why they switched to Gini. The simple answer is that the Entropy rule tends to produce even smaller terminal nodes ("end cut splits") and is usually less accurate than Gini. In our experience about one problem in twenty is best handled by the Entropy rule.

Class Probability:

The probability tree is a form of the Gini tree that deserves much more attention than it has received. Probability trees tend to be larger than Gini trees and the predictions made in individual terminal nodes tend to be less reliable, but the details of the data structure that they reveal can be very valuable.

When you are primarily interested in the performance of the top few nodes of a tree you should be looking at probability trees.

Twoing:

The major difference between the Twoing and other splitting rules is that Twoing tends to produce more balanced splits (in size). Twoing has a built-in penalty that makes it avoid unequal splits whereas other rules do not take split balance into account when searching for the best split. A Gini or Entropy tree could easily produce 90/10 splits whereas Twoing will tend to produce 50/50 splits. The differences between the Twoing and other rules become more evident when modeling multi-class targets with more than two levels. For example, if you were modeling segment membership for an eight-way segmentation, the Twoing and Gini rules would probably yield very different trees and performances.

Ordered Twoing:

The Ordered Twoing rule is useful when your target levels are ordered classes. For example, you might have customer satisfaction scores ranging from 1 to 5 and in your analysis you want to think of each score as a separate class rather than a simple score to be predicted by a regression. If you were to use the Gini rule CART would think of the numbers 1,2,3,4, and 5 as arbitrary labels without having any numeric significance. When you request Ordered Twoing you are telling CART that a “4” is more similar to a “5” than it is to a “1.” You can think of Ordered Twoing as developing a model that is somewhere between a classification and a regression.

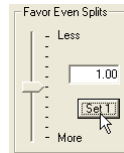
Ordered Twoing works by making splits that tend to keep the different levels of the target together in a natural way. Thus, we would favor a split that put the “1” and “2” levels together on one side of the tree and we would want to avoid splits that placed the “1” and “5” levels together. Remember that the other splitting rules would not care at all which levels were grouped together because they ignore the numeric significance of the class label.

As always, you can never be sure which method will work best. We have seen naturally ordered targets that were better modeled with the Gini method. You will need to experiment.

✎ Ordered Twoing works best with targets with numeric levels. When a target is a character variable, the ordering conducted by CART might not be to your liking. See the command reference manual section on the DISCRETE command for more useful information.

Favor Even Splits

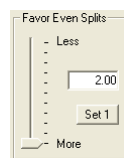
The “favor even splits” control is also on the Method tab and offers an important way to modify the action of the splitting rules. By default, the setting is 0, which indicates no bias in favor of even or uneven splits. In the display below we have set the splitting rule to Twoing and the “favor even splits” setting to 1.00.



The “favor even splits” control is set by the POWER parameter in the command language. For example, the command

```
METHOD TWOING, POWER=1
```

is how we would request the Twoing rule with a moderate favoring of even splits. Of course, you never have to deal with the command language if you do not want to, but knowing a little can be helpful. If you want to lean further in the direction of even splits then raise the setting to 2.00 as we do below:



The GUI limits your POWER setting to a maximum value of 2.00. This is to protect users from setting outlandish values. There are situations, however, in which a higher setting might be useful, and if so you will need to enter a command with a POWER setting of your choice. Using values greater than 5.00 is probably not helpful.

On binary targets when both “Favor even splits” and the unit cost matrix are set to 0, Gini, Symmetric Gini, Twoing, and Ordered Twoing will produce near identical results.

Although we make recommendations below as to which splitting rule is best suited to which type of problem, it is good practice to always use several splitting rules and compare the results. You should experiment with several different splitting rules and should expect different results from each. As you work with different types of data and problems, you will begin to learn which splitting rules typically work best for specific problem types. Nevertheless, you should never rely on a single rule alone; experimentation is always wise.

The following rules of thumb are based on our experience in the telecommunications, banking, and market research arenas, and may not apply to other subject areas. Nevertheless, they represent such a consistent set of empirical findings that we expect them to continue to hold in other domains and data sets more often than not.

For a two-level dependent variable that can be predicted with a relative error of less than 0.50, the Gini splitting rule is typically best.

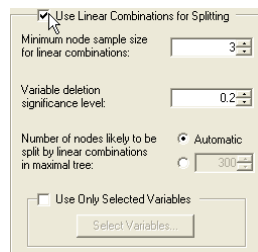
For a two-level dependent variable that can be predicted with a relative error of only 0.80 or higher, Power-Modified Twoing tends to perform best.

For target variables with four to nine levels, Twoing has a good chance of being the best splitting rule.

For higher-level categorical dependent variables with 10 or more levels, either Twoing or Power-Modified Twoing is often considerably more accurate than Gini.

Linear Combination Splits:

To deal more effectively with linear structure, CART has an option that allows node splits to be made on linear combinations of non-categorical variables. This option is implemented by clicking on the **Use Linear Combinations for Splitting** check box on the **Method** tab as seen below.



Minimum Node Sample Size

The minimum node sample size for linear combinations, which can be changed from the default of three by clicking the up or down arrows, specifies the minimum number of cases required in a node for linear combination splits to be considered. Nodes smaller than the specified size will be split on single variables only.

The default value is far too small for most practical applications. We would recommend using values such as 20, 50, 100 or more.

Variable Deletion Significance Level

The **Variable deletion significance level**, set by default at 0.20, governs the backwards deletion of variables in the linear combination stepwise algorithm. Using a larger setting will typically select linear combinations involving fewer variables. We often raise this threshold to 0.40 for this purpose.

Estimating Number of Linear Splits

By default, CART automatically estimates the maximum number of linear combination splits in the maximal tree. The automatic estimate may be overridden to allocate more linear combination workspace. To do so, click on the **Number of**

nodes likely to be split by linear combinations in maximal tree radio button and enter a positive value.

- 💡 CART will terminate the model-building process prematurely if it finds that it needs more linear combination splits than were actually reserved.
- 💡 Linear combination splits will be automatically turned off for all nodes that have any constant predictors (all values the same for all records). Thus, having a constant predictor in the training data will effectively turn off linear combinations for the entire tree.
- 📝 Command-line users will use the following command syntax to specify linear combinations.

```
LINEAR N=<min_cases>, DELETE=<signif_level>, SPLITS=<max_splits>
```

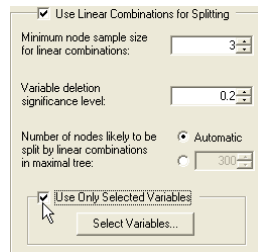
LC Lists: Use Only Selected Variables

LC lists are a new addition to CART and can radically improve the predictive power and intuitive usefulness of your trees. In legacy CART if you request a search for linear combination splitters ALL the numeric variables in your predictor (KEEP) list are eligible to enter the linear combination (LC). In every node with a large enough sample size CART will look for the best possible LC regardless of which variables combine to produce that LC.

We have found it helpful to impose some structure on this process by allowing you to organize variables into groups from which LCs can be constructed. If you create such groups, then any LC must be constructed entirely from variables found in a single group. In a biomedical study you might consider grouping variables into demographics such as AGE and RACE, lifestyle or behavioral variables such as SMOKE and FTV, and medical history and medical condition variables such as UI, PTD, and LWT. Specifying LCLISTS in this way will limit any LCs constructed to those that can be created from the variables in a *single* list.

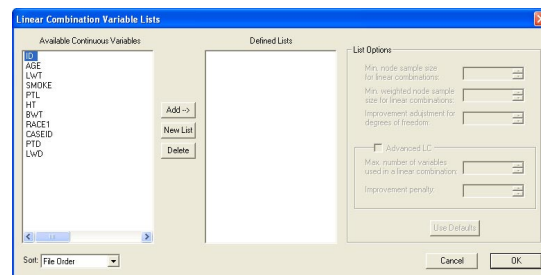
- 🔗 Time series analysts can create one LCLIST for each predictor and its lagged values. LCs constructed from such a list can be thought of as distributed lag predictors.
- 🔗 A variable can appear on more than one LCLIST, meaning that LC lists can overlap. You can even create an LCLIST with all numeric variables on it if you wish.

Below we have checked the box that activates LC lists for our example:



Clicking on the **[Select Variables]** button brings up this new window in which you may create your LC lists.

✂ Only numeric variables will be displayed in this window. Categorical variables will not be considered for incorporation into an LC even if they are simple 0/1 indicators. This is one good reason to treat your 0/1 indicators as numeric rather than categorical predictors.



Click on **New List** to get started and then select the variables you want to include in the first list. We will select AGE and SMOKE. **Add** them and then click again on **New List** to start a second list. Now **Add** HT, PTD, LWD and click OK to complete the LCLIST setup. Click **Start** to begin the run.

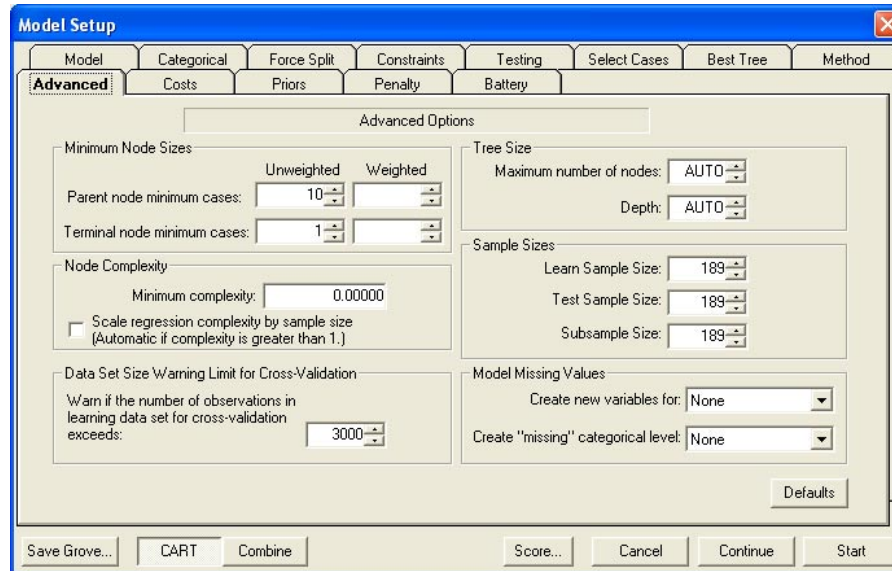
Hovering your mouse over the nodes of this tree will allow you to quickly spot where linear combination splits have been found. Here we double click on the root node of the navigator to bring up this display.

Observe that the node is split on a linear combination of the two variables AGE and SMOKE with the splitter displayed near the top of the window. The improvement score of this LC is .0433, which is about 20% better than the best single-variable splitter PTD, which has an improvement score of .0355.

If you do not restrict the LCs with LCLISTS and instead run a legacy CART with linear combinations, you won't find any LCs reported. This is not a surprise; we have found it many times. Limiting LCs to a few choice variables is likely to yield better results than allowing CART to search over all available variables, a reflection of the fact that the LC search procedure cannot guarantee a global maximum.

The Advanced Tab

The **Model Setup—Advanced** tab allows you to specify additional tree-building control options and settings. You should not hesitate to learn the meaning and use of these controls, as they can be key to getting the best results.



Parent node minimum cases (ATOM)

When do we admit that we do not have enough data to continue? Theoretically, we can continue splitting nodes until we run out of data, for example, when there is only one record left in a node. In practice it makes sense to stop tree growing when the sample size is so small that no one would take the split results seriously. The default setting for the smallest node we consider splitting is 10, but we frequently set the minimum to 20, 50, 100 or even 200 in very large samples.

Terminal node minimum sizes (MINCHILD)

This control specifies the smallest number of observations that may be separated into a child node. A large node might theoretically be split by placing one record in one child node and all other records into the other node. However, such a split would be rightfully regarded as unsatisfactory in most instances. The MINCHILD control allows you to specify a smallest child node, below which no nodes can be constructed. Naturally, if you set the value too high you will prevent the construction of any useful tree.

- ✎ Increasing allowable parent and child node sizes enables you to both control tree growth and to potentially fit larger problems into limited workspace (RAM).

- ✂ You will certainly want to override the default settings when dealing with large datasets.
- ✂ The parent node limit (ATOM) must be at least twice the terminal node (MINCHILD) limit and otherwise will be adjusted by CART to comply with the parent limit setting.
- ✂ We recommend that ATOM be set to at least three times MINCHILD to allow CART to consider a reasonable number of alternative splitters near the bottom of the tree. If ATOM is only twice MINCHILD then a node that is just barely large enough to be split can be split only into two equal-sized children.
- 📄 Command-line users will use the following command syntax to specify node limitations.

```
LIMIT ATOM=<parent limit>, MINCHILD=<child limit>
```

Minimum complexity

This is a truly advanced setting with no good short explanation for what it means, but you can quickly learn how to use it to best limit the growth of potentially large trees. The default setting of zero allows the tree-growing process to proceed until the “bitter end.” Setting complexity to a value greater than zero places a penalty on larger trees, and causes CART to stop its tree-growing process before reaching the largest possible tree size. When CART reaches a tree size with a complexity parameter equal to or smaller than your pre-specified value, it stops the tree-splitting process on that branch. If the complexity parameter is judiciously selected, you can save computer time and fit larger problems into your available workspace. (See the main reference manual for guidance on selecting a suitable complexity parameter.)

- ✂ As described in detail in the main reference manual, check the **Complexity Parameter** column in the TREE SEQUENCE section of the CART Output to get the initial feel for which complexity values are applicable for your problem.

The **Scale Regression** check box specifies that, for a regression problem, the complexity parameter should be scaled up by the learn-sample size.

- 📄 Command-line users will use the following command syntax to specify this complexity parameter.

```
BOPTIONS COMPLEXITY = <value>, [SCALED]
```

Dataset Size Warning Limit for Cross Validation

By default, 3,000 is the maximum number of cases allowed in the learning sample before cross validation is disallowed and a test sample is required. To use cross validation on a file containing more than 3,000 records, increase the value in this box to at least the number of records in your data file.



Command-line users will use the following command syntax.

```
BOPTIONS CVLEARN = <N>
```

Maximum number of nodes (NODES)

Allows you to specify a maximum allowable number of nodes in the largest tree grown. If you do not specify a limit CART may allow as many as one terminal node per data record. When a limit on NODES is specified the tree generation process will stop when the maximum allowable number of nodes (internal plus terminal) is reached. This is a crude but effective way to limit tree size.

Depth

This setting limits the tree growing to a maximum depth. The root node corresponds to the depth of zero. Limiting a tree in this way is likely to yield an almost perfectly balanced tree with every branch reaching the same depth. While this may appeal to your aesthetic sensibility it is unlikely to be the best tree for predictive purposes.

By default CART sets the maximum DEPTH value so large that it will never be reached.



Unlike complexity, these NODES and DEPTH controls may handicap the tree and result in inferior performance.



Some decision tree vendors set depth values to small limits such as five or eight. These limits are generally set very low to create the illusion of fast data processing. If you want to be sure to get the best tree you need to allow for somewhat deeper trees.



Command-line users will use the following command syntax.

```
LIMIT NODES = <N>, DEPTH = <N>
```

Learn Sample Size (LEARN)

The LEARN setting limits CART to processing only the first part of the data available and simply ignoring any data that comes after the allowed records. This is useful when you have very large files and want to explore models based on a small portion of the initial data. The control allows for faster processing of the data because the entire data file is never read.

Test Sample Size

The TEST setting is similar to LEARN: it limits the test sample to no more than the specified number of records for testing. The test records are taken on a first-come-first served basis from the beginning of the file. Once the TEST limit is reached no additional test data are processed.

Sub-sample Size

Node sub-sampling is an interesting approach to handling very large data sets and also serves as a vehicle for exploring model sensitivity to sampling variation. Although node sub-sampling was introduced in the first release of the CART mainframe software in 1987, we have not found any discussion of the topic in the scientific literature. We offer a brief discussion here.

Node sub-sampling is a special form of sampling that is triggered for special purposes during the construction of the tree. In node sub-sampling the analysis data are *not* sampled. Instead we work with the complete analysis data set. When node sub-sampling is turned on we conduct the process of searching for a best splitter for a node on a subsample of the data in the node. For example, suppose our analysis data set contained 100,000 records and our node sub-sampling parameter was set to 5,000. In the root node we would take our 100,000 records and extract a random sample of 5,000. The search for the best splitter would be conducted on the 5,000 random record extract. Once found, the splitter would be applied to the full analysis data set. Suppose this splitter divided the 100,000 root node into 55,000 records on the left and 45,000 records on the right. We would then repeat the process of selecting 5,000 records at random in each of these child nodes to find their best splitters.

As you can see, the tree generation process continues to work with the complete data set in all respects except for the split search procedure. By electing to use node sub-sampling we create a shortcut for split finding that can materially speed up the tree-growing process.

But is node sub-sampling a good idea? That will depend in part on how rare the target class of interest is. If the 100,000 record data set contains only 1,000 YES records and 99,000 NO records, then any form of sub-sampling is probably not helpful. In a more balanced data set the cost of an abbreviated split search might be minimal and it is even possible that the final tree will perform better. Since we cannot tell without trial and error we would recommend that you explore the impact of node sub-sampling if you are inclined to consider this approach.



Command-line users will use the following command syntax.

```
LIMIT LEARN = <N>, TEST = <N>, SUBSAMPLE = <N>
```

Model Missing Values

CART 6.0 introduces a new set of missing value analysis tools for automatic exploration of the optimal handling of your incomplete (missing) data. On request, CART will automatically add missing value indicator variables (MVs) to your list of predictors and conduct a variety of analyses using them. For a variable named X1, the MVI will be named X1_MIS and coded as 1 for every row with a missing value for X1 and 0 otherwise. If you activate this control, the MVIs will be created automatically

(as temporary variables) and will be used in the CART tree if they have sufficient predictive power. MVIs allow formal testing of the core predictive value of knowing that a field is missing.

Create new variable for (MVI)

There are three control options for missing values indicators. The user can request MVIs for all variables, or limit them to either continuous only, or categorical only predictor variables.



Command-line users will use the following command syntax.

The following command syntax will turn on MVIs for all variables.

```
BOPTIONS MISSING = YES
```

To limit MVIs to categorical (discrete) variables only we use:

```
BOPTIONS MISSING = DISCRETE
```

To limit MVIs to continuous variables only we use:

```
BOPTIONS MISSING = CONTINUOUS
```

Create "missing" categorical level

For categorical variables an MVI can be accommodated in two ways: by adding a separate MVI variable as show above, or by treating missing as a valid "level."

The **Create "missing" categorical level** control specifies whether missing values for discrete variables are treated as truly MISSING or are considered a legal and distinct level. The user can choose from three control options.

1. Process missing values for ALL variables as legal.
2. Process missing values only for predictor variables as legal.
3. Process missing values only for the target variable as legal.



Command-line users will use the following command syntax.

The following command syntax will process missing values for all variables as legal.

```
DISCRETE MISSING = ALL
```

To process missing values only for predictor variables as legal:

```
DISCRETE MISSING = LEGAL
```

To process missing values only for the target variable as legal:

```
DISCRETE MISSING = TARGET
```

To process missing values as truly missing (default setting):

```
DISCRETE MISSING = MISSING
```

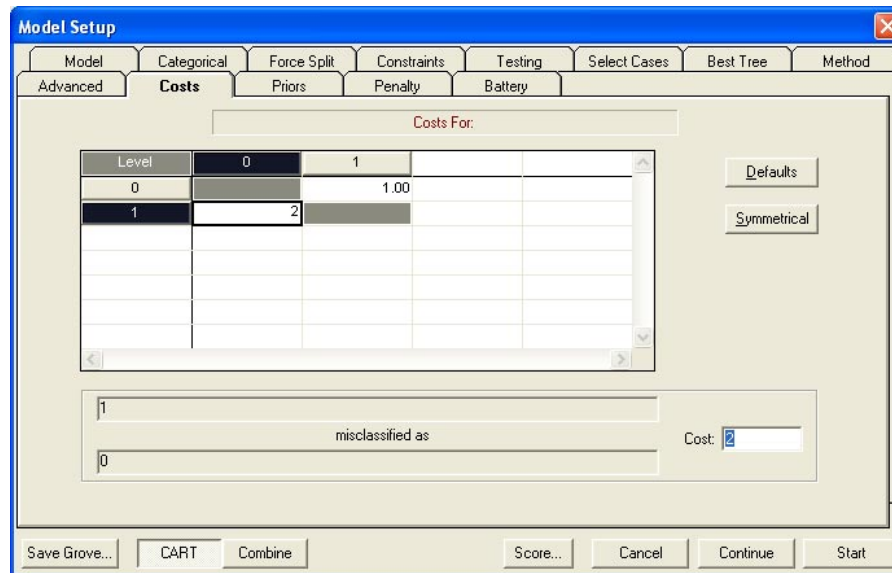
The Cost Tab

Because not all mistakes are equally serious or equally costly, decision makers are constantly weighing quite different costs. If a direct mail marketer sends a flyer to a person who is uninterested in the offer the marketer may waste \$1.00. If the same marketer fails to mail to a would-be customer, the loss due to the foregone sale might be \$50.00. A false positive on a medical test might cause additional more costly tests amounting to several hundreds of dollars. A false negative might allow a potentially life-threatening illness to go untreated. In data mining, costs can be handled in two ways:

- ✂ on a post-analysis basis where costs are considered after a cost-agnostic model has been built, and
- ✂ on a during-analysis basis in which costs are allowed to influence the details of the model.

CART is unique in allowing you to incorporate costs into your analysis and decision making using either of these two strategies.

To incorporate costs of mistakes directly into your CART tree, complete the matrix in the **Model Setup—Cost** tab illustrated below. For example, if misclassifying low birth weight babies (LOW=1) is more costly than misclassifying babies who are not low birth weight (LOW=0), you may want to assign a penalty of two to misclassifying class 1 as 0. (See the main reference manual for a detailed discussion of misclassification costs.)



- ✂ Only cell ratios matter, that is, the actual value in each cell of the cost matrix is of no consequence—setting costs to 1 and 2 for the binary case is equivalent to setting costs to 10 and 20.
- ✂ In a two-class problem, set the lower cost to 1.00 and then set the higher cost as needed. You may find that a small change in a cost is all that is needed to obtain the balance of correct and incorrect and the classifications you are looking for. Even if one cost is 50 times greater than another, using a setting like 2 or 3 may be adequate.
- ✂ On binary classification problems, manipulating costs is equivalent to manipulating priors and vice versa. On multilevel problems, however, costs provide more detailed control over various misclassifications than do priors.

By default, all costs are set to one (unit costs).

To change costs anywhere in the matrix, click on the cell you wish to alter and enter a positive numeric value in the text box called Cost. To specify a symmetrical cost matrix, enter the costs in the upper right triangle of the cost matrix and click on **[Symmetrical]**. CART automatically updates the remaining cells with symmetrical costs. Click **[Defaults]** to restore to the unit costs.

- 📄 Command-line users should use the following command syntax for each cell that has a non-unit value.

```
MISCLASSIFY COST=<value> CLASSIFY <origin_class> AS <predicted>
-----
MISCLASSIFY COST = 2 CLASSIFY 1 AS 0
```

- 💡 CART requires all costs to be strictly positive (zero is not allowed). Use small values, such as .001, to effectively impose zero costs in some cells.
- 🔗 We recommend conducting your analyses with the default costs until you have acquired a good understanding of the data from a cost-neutral perspective.

The Priors tab

The **Model Setup—Priors** tab is one of the most important options you can set in shaping a classification analysis and you need to understand the basics to get the most out of CART. Although the PRIORS terminology is unfamiliar to most analysts the core concepts are relatively easy to grasp. Market researchers and biomedical analysts make use of the priors concepts routinely but in the context of a different vocabulary.

We start by discussing a straightforward 0/1 or YES/NO classification problem. In most real world situations, the YES or 1 group is relatively rare. For example, in a large field of prospects only a few become customers, relatively few borrowers default on their loans, only a tiny fraction of credit card transactions and insurance claims are fraudulent, etc. The relative rarity of a class in the real world is usually reflected in the data available for analysis. A file containing data on 100,000 borrowers might include no more than 4,000 bankrupts for a mainstream lender.

Such unbalanced data sets are quite natural for CART and pose no special problems for analysis. This is one of CART's great strengths and differentiates CART from other analytical tools that do not perform well unless the data are "balanced."

The CART default method for dealing with unbalanced data is to conduct all analyses using measures that are relative to each class. In our example of 100,000 records containing 4,000 bankrupts, we will always work with ratios that are computed relative to 4,000 for the bankrupts and relative to 96,000 for the non-bankrupts. By doing everything in relative terms we bypass completely the fact that one of the two groups is 24 times the size of the other.

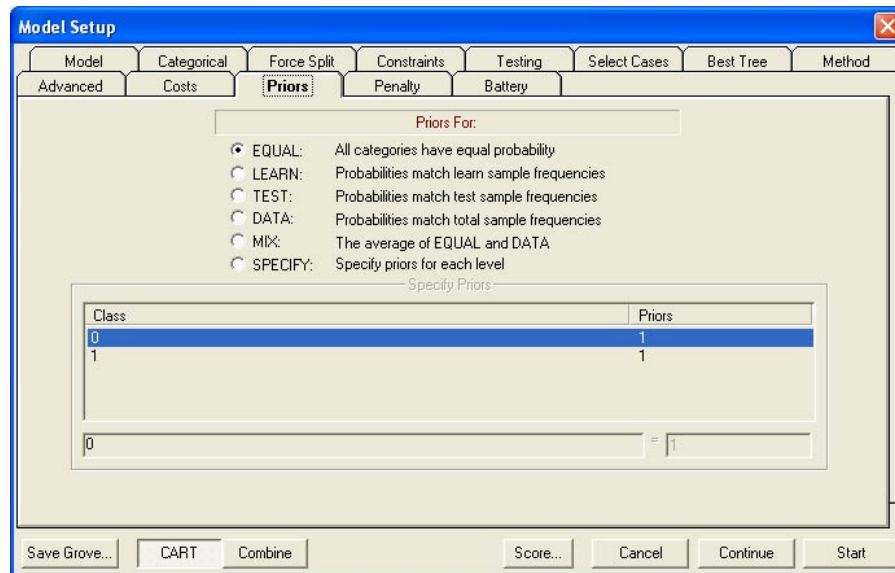
This method of bookkeeping is known as PRIORS EQUAL. It is the default method used for classification trees and often works supremely well. It is the setting we almost always use to start our exploration of new data. This default setting frequently gives the most satisfactory results because each class is treated as equally important for the purpose of achieving classification accuracy.

Priors are usually specified as fractions that sum to 1.0. In a two-class problem EQUAL priors would be expressed numerically as 0.50, 0.50, and in a three-class problem they would be expressed as 0.333, 0.333, 0.333.

- PRIORS may look like weights but they are not weights. Priors reflect the relative size of a class after CART has made its adjustments. Thus, PRIORS EQUAL assures that no matter how small a class may be relative to the other classes, it will be treated as if it were of equal size.
- ✂ PRIORS DATA (or PRIORS LEARN or PRIORS TEST) makes no adjustments for relative class sizes. Under this setting small classes will have less influence on the CART tree and may even be ignored if they interfere with CART's ability to classify the larger classes accurately.
- ✂ PRIORS DATA is perfectly reasonable when the importance of classification accuracy is proportional to class size. Consider a model intended to predict which political party will be voted for with the alternatives of Conservative, Liberal, Fringe1 and Fringe2. If the fringe parties together are expected to represent about 5% of the vote, an analyst might do better with PRIORS DATA, allowing CART to focus on the two main parties for achieving classification accuracy.

Six different priors options are available, as follows:

EQUAL	Equivalent to weighting classes to achieve BALANCE (default setting)
DATA	Larger classes are allowed to dominate the analysis
MIX	Priors set to the average of the DATA and EQUAL options
LEARN	Class sizes calculated from LEARN sample only
TEST	Class sizes calculated from TEST sample only
SPECIFY	Priors set to user-specified values



Default Priors settings: priors equal (applicable to classification trees only).

You can change the priors setting by clicking on the new setting's radio button. If you select SPECIFY, you must also enter a value for each level of your target variable. Simply highlight the corresponding class and type in the new value.

✍ Only the ratios of priors matter—internally, CART normalizes the specified priors so that the values always sum to one.

💡 Certain combinations of priors may result in a “No Tree Built” situation. This means that, according to this set of priors, having no tree (a trivial model, which makes the same class assignment everywhere) is no worse than having a tree. Knowing that your target cannot be predicted from your data can be very valuable and in some cases is a conclusion you were looking for.

📄 From the Command-line use the following syntax.

```
PRIORS EQUAL
PRIORS DATA
PRIORS MIX
PRIORS LEARN
PRIORS TEST
PRIORS SPECIFY <class1>=<value1>, <class2>=<value2>, ... etc.
-----
PRIORS SPECIFY 0 = .25, 1 = .75
```

- If the target variable contains >5000 values, you must use the command line for user-specified priors.

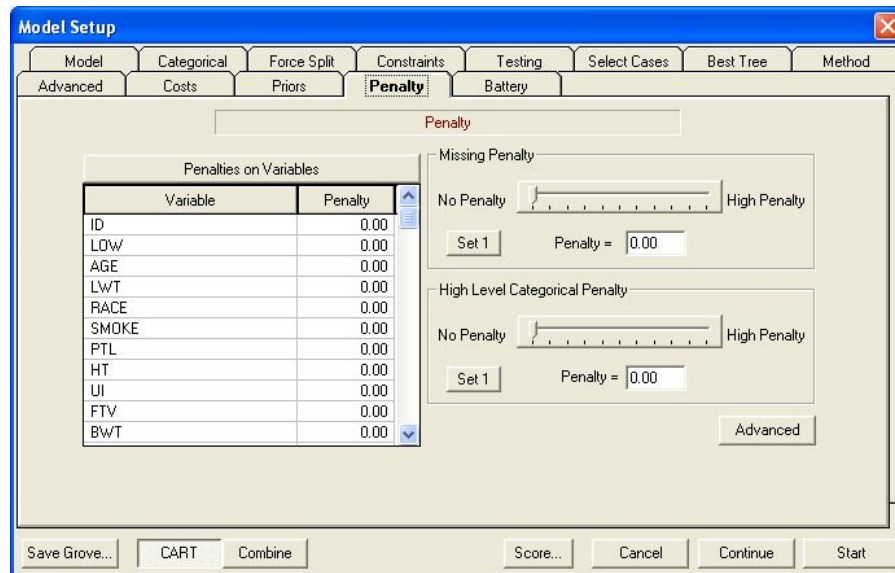
The Penalty tab

The penalties available in CART were introduced by Salford Systems starting in 1997 and represent important extensions to decision tree technology. Penalties can be imposed on variables to reflect a reluctance to use a variable as a splitter. Of course, the modeler can always exclude a variable; the penalty offers an opportunity to permit a variable into the tree but only under special circumstances. The three categories of penalty are:

- ✂ Missing Value Penalty: Predictors are penalized to reflect how frequently they are missing. The penalty is recalculated for every node in the tree.
- ✂ High Level Categorical Penalty: Categorical predictors with many levels can distort a tree due to their explosive splitting power. The HLC penalty levels the playing field.
- ✂ Predictor Specific Penalties: Each predictor can be assigned a custom penalty.

A penalty will lower a predictor's improvement score, thus making it less likely to be chosen as the primary splitter. These penalties are defined in the **Model Setup—Penalty** tab. Penalties specific to particular predictors are entered in the left panel next to the predictor name and may range from zero to one inclusive.

Penalties for missing values (for categorical and continuous predictors) and a high number of levels (for categorical predictors only) can range from "No Penalty" to "High Penalty" and are normally set via the slider on the Penalty tab, as seen in the following illustration.



In the screen we have set both the Missing Values and the HLC penalties to the frequently useful values of 1.00. Advanced users wishing control over the missing value and high-level categorical penalty details can click the **[Advanced]** button.

Penalties on Variables

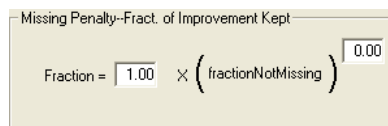
The penalty specified is the amount by which the variable's improvement score is reduced before deciding on the best splitter in a node. Imposing a 0.10 penalty on a variable will reduce its improvement score by 10%. You can think of the penalty as a "handicap": with a 0.10 penalty we are saying that the penalized variable must be at least 10% better than any other variable to qualify as the splitter.

- ✎ Penalties may be placed to reflect how costly it is to acquire data. For example, in database and targeted marketing, selected data maybe available only by purchase from specialized vendors. By penalizing such variables we make it more difficult for such variables to enter the tree, but they will enter when they are considerably better than any alternative predictor.
- ✎ Predictor specific penalties have been used effectively in medical diagnosis and triage models. Predictors that are "expensive" because they require costly diagnostics, such as CT scans, or that can only be obtained after a long wait (say 48 hours for the lab results), or that involve procedures that are unpleasant for the patient, can be penalized. If penalizing these variables leads to models that are only slightly less predictive, the penalties help physicians to optimize diagnostic procedures.

- ✎ Setting the penalty to one is equivalent to effectively removing this predictor from the predictor list.

Missing Values Penalty

At every node every predictor competes to be the primary splitter. The predictor having the best improvement score is selected to be the primary splitter. Variables with no missing values have their improvement scores computed using all the data in the node, while variables with missings have their improvement scores calculated using only the subset with complete data. Since it is easier to be a good splitter on a small number of records this tends to give heavily missing variables an advantage. To level the playing field, variables can be penalized in proportion to the degree to which they are missing. This proportion missing is calculated separately at each node in the tree. For example, a variable with good data for only 30% of the records in a node would receive only 30% of its calculated improvement score. In contrast, a variable with good data for 80% of the records in a node would receive 80% of its improvement score. A more complex formula is available for finer control over the missing value penalty using the "Advanced" version of the **Penalty** tab.



Missing Penalty-Fract. of Improvement Kept

Fraction = 1.00 × (fractionNotMissing)^{0.00}

Suppose you want to penalize a variable with 70% missing data very heavily, while barely penalizing a variable with only 10% missing data. The advanced tab lets you do this by setting a fractional power on the percent of good data. For example, using the square root of the fraction of good data to calculate the improvement factor would give the first variable (with 70% missing) a .55 factor and the second variable (with 10% missing) a .95 factor.

The expression used to scale improvement scores is:

$$S = a * (\text{proportion_not_missing})^b$$

The default settings of $a = 1$, $b = 0$ disable the penalty entirely; every variable receives a factor of 1.0. Useful penalty settings set $a = 1$ with $b = 1.00$, or 0.50. The closer b gets to 0 the smaller the penalty. The fraction of the improvement kept for a variable is illustrated in the following table, where "%good" = the fraction of observations with non-missing data for the predictor.

%good	b=.75	b=.50
0.9	0.92402108	0.948683298
0.8	0.84589701	0.894427191
0.7	0.76528558	0.836660027
0.6	0.68173162	0.774596669
0.5	0.59460355	0.707106781
0.4	0.50297337	0.632455532
0.3	0.40536004	0.547722558
0.2	0.29906975	0.447213595
0.1	0.17782794	0.316227766

Looking at the bottom row of this table we see that if a variable is only good in 10% of the data it would receive 10% credit if $b=1$, 17.78% credit if $b=.75$, and 31.62% credit if $b=.50$. If $b=0$, the variable would receive 100% credit because we would be ignoring its degree of missingness.

✂ In most analyses we find that the overall predictive power of a tree is unaffected by the precise setting of the missing value penalty. However, without any missing value penalty you might find heavily missing variables appearing high up in the tree. The missing value penalty thus helps generate trees that are more appealing to decision makers.

High-level Categorical Penalty


Categorical predictors present a special challenge to decision trees. Because a 32-level categorical predictor can split a data set in over two billion ways, even a totally random variable has a high probability of becoming the primary splitter in many nodes. Such spurious splits will not prevent CART from eventually detecting the true data structure in large data sets, but they make the process inefficient. First, they add unwanted nodes to a tree, and as they promote the fragmentation of the data into added nodes, the reduced sample size as we progress down the tree makes it harder to find the best splits.

To protect against this possibility CART offers a high-level categorical predictor penalty used to reduce the measured splitting power. On the "Basic" Penalty dialog, this is controlled with a simple slider.

The "Advanced" Penalty dialog allows access to the full penalty expression. The improvement factor is expressed as:


$$S = \min \left\{ 1, 1 + c * \left(\left(\frac{\log_2[node_size]}{N_categories - 1} \right)^d - 1 \right) \right\}$$


By default, $c = 1$ and $d = 0$; these values disable the penalty. We recommend that the categorical variable penalty be set to ($c = 1$, $d = 1$), which ensures that a categorical predictor has no inherent advantage over a continuous variable with unique values for every record.

 Command-line users will use the following command syntax to specify variable penalties.


```
PENALTY <var>=<penalty>, /MISSING=<mis_val1>,<mis_val2>,
                                HLC=<hlc_val1>,<hlc_val2>

PENALTY /MISSING=1,1, HLC=1,1
```

 The missing value and HLC penalties apply uniformly for all variables. You cannot set different HLC or missing value penalties to different variables. You choose one setting for each penalty and it will apply to all variables.

 You can set variable specific penalties *and* general missing value and HLC penalties. Thus, if you have a categorical variable Z that is also sometimes missing you could have all three penalties applying to this variable at the same time.

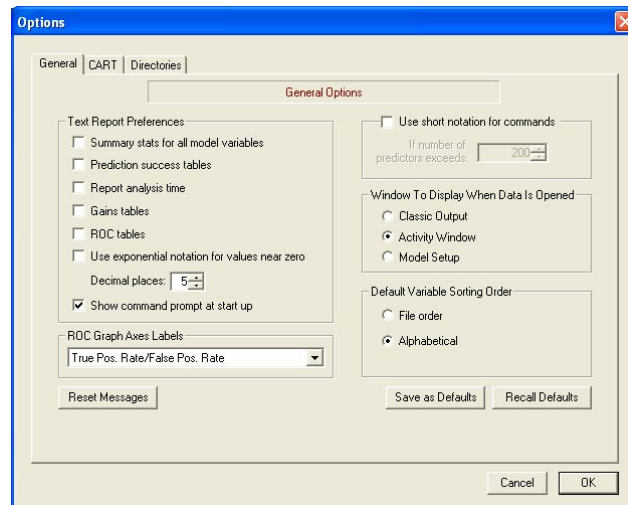
Setting Reporting, Random Number and Directory Options

This section is a guide to the reporting and other fine-tuning global controls you may want to set before you grow your trees. These parameters are contained in the Options dialog accessed by selecting **Options...** from the **Edit** menu (or clicking , on the toolbar icon).

If you are in the Model Setup dialog box, you must first click on the **[Continue]** button to access **Options** from the **Edit** menu.

General Text Report Preferences

CART is actually part of an integrated data mining system offering several analytical methods. The CART 6.0 -Standard Edition product offers only the CART subsystem at this time but in the future other modules will become available. The **Options—General** tab controls report and display preferences that are common across several data mining technologies (including TreeNet and RandomForests). The screen shot below shows one set of user preferences:

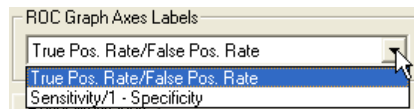


The report preferences allow you to turn on and off the following parts in the CART classic output (with command-line equivalents included):

- ◆ Summary stats for all model variables—mean, standard deviation, min, max, etc. In classification models the stats are reported for the overall train and test samples and then separately for each level of the target.
LOPTIONS MEANS=YES | NO
- ◆ Prediction success tables - confusion matrix with misclassification counts and %'s by class level.
LOPTIONS PREDICTIONS=YES | NO
- ◆ Report analysis time - CPU time required for each stage of the analysis.
LOPTIONS TIING=YES | NO
- ◆ Report Gains tables.
LOPTIONS GAINS=YES | NO
- ◆ Report ROC tables.
LOPTIONS ROC=YES | NO
- ◆ Decimal places - precision to which the numerical output is printed.
FORMAT = <N>
- ◆ Exponential notation for near-zero values - exponential notation used for values close to zero.
FORMAT = <N> / UNDERFLOW

ROC Graph Labels

ROC graphs are traditionally labeled differently in different industries. You can select from the two labeling schemes displayed below:



Press the **[Save as Defaults]** button to save your preferences permanently. If you have made some temporary changes and wish to restore your previously-saved defaults, press the **[Recall Defaults]** button.

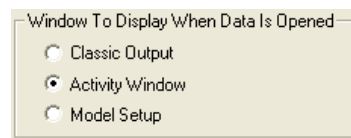
Use Short Command Notation

Sets the minimal number of predictors that triggers a short command notation in the command log. When the number of predictors is small, each predictor is printed in the command log (for example, KEEP or CATEGORY commands). However, when the number of predictors exceeds the limit, CART uses “dash” convention to indicate ranges of predictors (for example, X1-X5).

✎ This setting only affects the GUI logging mechanism. The command parser supports both short and standard command notations.

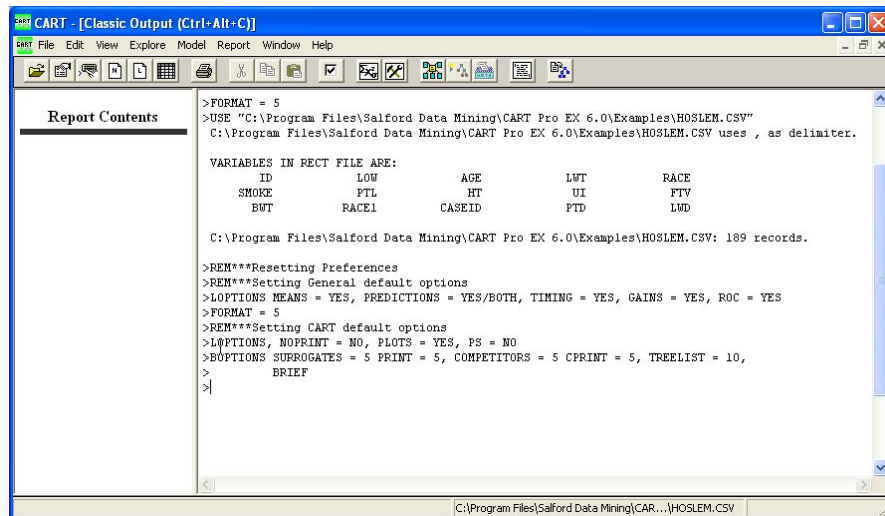
Window to Display When File Is Opened

When you open a data file CART gives you three choices for what to do next:



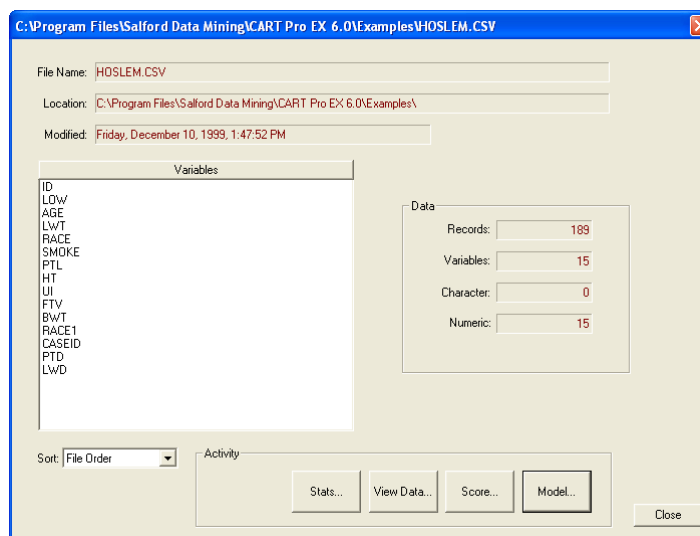
Classic Output

This is the classic text mainframe style output suitable for diehard UNIX and Linux gurus. You will be greeted with a plain text screen looking something like:




Data Description/Activity Window

This new window can function as a brief description of your data file and a control panel for other data exploration and analysis activities.

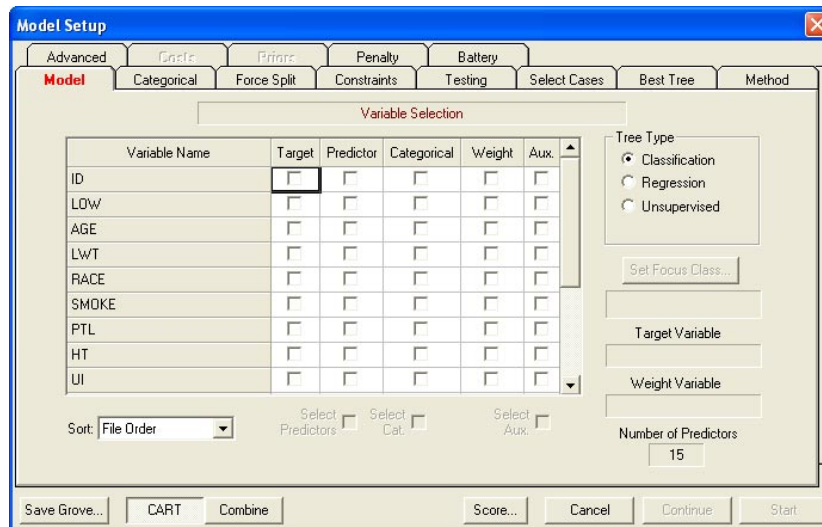


From this screen you can conveniently request summary statistics, a spreadsheet view of the data, or the model set-up dialog, and you can also move directly to scoring the data using a previously-saved model.

Once you close this window it can be reopened by clicking on the  toolbar icon (hammer and wrench icon).

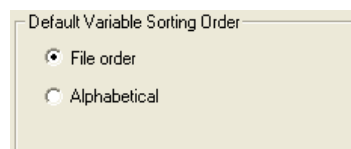
Model Setup

This is the window that came up automatically in CART 4.0 and CART 5.0 and you can also put CART 6.0 into this mode.



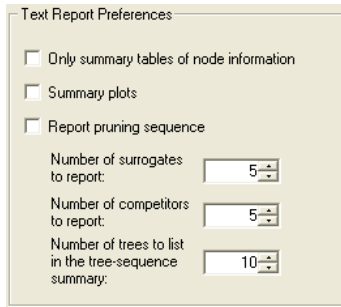
Default Variable Sorting Order

Many GUI displays include a list of variables and you can always change the sort order between **Alphabetical** and **File Order** (the order in which the variables appear in your data file). This setting allows you to determine the ordering that will always show first when a dialog is opened.



Controlling CART Report Details

The parameters controlling the contents of the CART Output window can be set in the **Options—CART** tab. This is the middle tab on the **Options** dialog. The default Reporting settings are shown below:



Full Node Detail or Summaries Only

Previous versions of CART printed full node detail for CART trees. These reports can be voluminous as they contain about one text page for every node in an optimal tree. If you elect to produce these details you can easily end up with more than the equivalent of 1000 pages of plain text reports.

We have now set the default to printing only summary tables, as most users do not refer to the classic text node detail.

- ✎ You can always recover the full node detail text report from any saved grove file via the TRANSLATE facility. Thus, there is no longer any real need to produce this text during the normal tree-growing process.

Summary Plots

These are classic mainframe line printer style plots for a few classic CART graphs. You can see these plots in the GUI so they are turned **off** by default.

Number of Surrogates to Report

Sets the maximum number of surrogates that can appear in the text report and the navigator displays.

- ✎ This setting only affects the displays in the text report and the Navigator windows. It does not affect the number of surrogates calculated.

The maximum number of surrogates calculated is set in the Best Tree tab of the Model Setup dialog.


You can elect to try to calculate 10 surrogate splitters for each node but then display only the top five. No matter how many surrogates you request you will get only as many as CART can find. In some nodes there are no surrogates found and the displays will be empty.

- 📄 The command-line equivalent of the number of surrogates to **report** is:


```
BOPTIONS PRINT=<N>
```


Number of Competitors to Report

Sets the maximum number of competitors that appear in reports.

 Every variable specified in your KEEP list or checked off as an allowed predictor on your Model Set Up is a competitor splitter. Normally we do not want or need to see how every one of them performed. The default setting displays the top five but there is certainly no harm in setting this number to a much larger value.


CART tests every allowed variable in its search for the best splitter. This means that CART always measures the splitting power of every predictor in every node. You only need to choose how much of this information you would like to be able to see in a navigator. Choosing a large number can increase the size of saved navigators/groves.


 Command-line equivalent

```
BOPTIONS COMPETITORS=<N>
```

Number of Trees to List in the Tree Sequence Summary

Each CART run prints a summary of the nested sequence of trees generated during growing and pruning. The number of trees listed in the tree-sequence summary can be increased or decreased from the default setting of 10 by entering a new value in the text box.

 This option only affects CART's classic output.

 Command-line equivalent

```
BOPTIONS TREELIST=<N>
```

Cross-validation Details: Classic Text Report

If you use the cross-validation testing method, you can request a text report for each of the maximal trees generated in each cross-validation run by clicking on the corresponding radio button for this option.

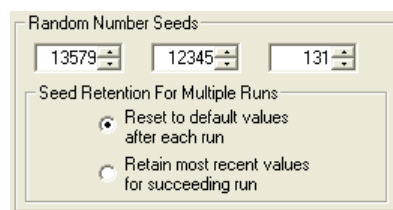
For example, if testing is set to the default 10-fold cross validation, a report for each of the ten cross-validated trees will follow the report on the final pruned tree in the text output. For this option to have full effect be sure to uncheck the "Only summary tables of node information." The GUI offers more a convenient way to review these CV details.

Command-line equivalent

BOPTIONS BRIEF
BOPTIONS COPIOUS

Controlling Random-Number Seed Values

As illustrated below, the **Options—CART** tab also allows you to set the random-number seed and to specify whether the seed is to remain in effect after a tree is built or data are dropped down a tree. Normally the seed is reset to 13579, 12345, and 131 on start-up and after each tree is constructed or after data are dropped down a tree. The seed will retain its latest value after the tree is built if you click on the **Retain most recent values for succeeding run** radio button.




Command-line equivalent.

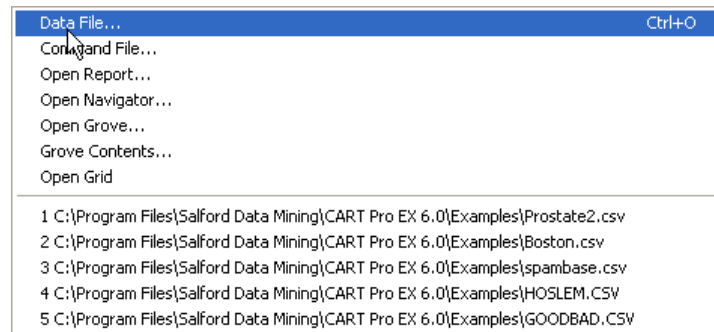
SEED <N1>, <N2>, <N3>, NORETAIN
SEED <N1>, <N2>, <N3>, RETAIN

Setting Directory Preferences

The **Option—Directories** tab allows you to set default directory preferences for input (data, model and command), output (model, scoring results, translation code and text report), and temporary files. By default, all input and output directories are initially set to the CART installation directory; the temporary directory is your machine's temporary Windows directory. Below we have set directory preferences for our input and output files.

To change any of the default directories, click on the  button next to the appropriate directory and specify a new directory in the **Select Default Directory** dialog box. CART will retain default directory settings in subsequent analysis sessions.

When the **Most Recently Used File** list checkbox is marked, CART adds the list of recently-used files to the **File->Open** menu.



Input Files

- Data: –input data sets (train and test) for modeling
- Model information: –previously-saved model files (navigators and groves)
- Command: –command files

Output Files

- Model information: –model files (groves) will be saved here
- Prediction results: –output data sets from scoring and translation code
- Run report: –classic output

Temporary Files

- Temporary: –where CART will write temporary work files as needed
- where CART will write the command log audit trail

✂ We suggest dedicating a separate temporary folder to CART.

Make it a habit to routinely check the Temporary Files Directory for unwanted scratch files. These should only appear if for some reason your system crashed or was powered down in a way that did not permit CART to clean up.

✂ Depending on your preferences, you may choose one of two working styles:

- (1) using the same location for both input and output files
- (2) using separate locations for input and output files

✂ The files with names like CART06125699_.txt are valuable records of your work sessions and provide an audit trail of your modeling activity. Think of them as emergency copies of your command log. You can delete these files if you are confident that your other records are adequate.

💡 Make sure that the drive where the temporary folder is located will have enough space (at least the size of the largest data set you are planning to use).

Additional Control Functions



–Control icon that automatically copies your **Data** file info to all other locations in the dialog (except the Temporary File location).



–Control icon that lets the user browse among directories.



–Control that allows the user to select from a list of previously-specified directories.



–Control that allows the user to specify how many recently-used files to remember in the **File-Open** menu. The maximum allowed is 20 files.

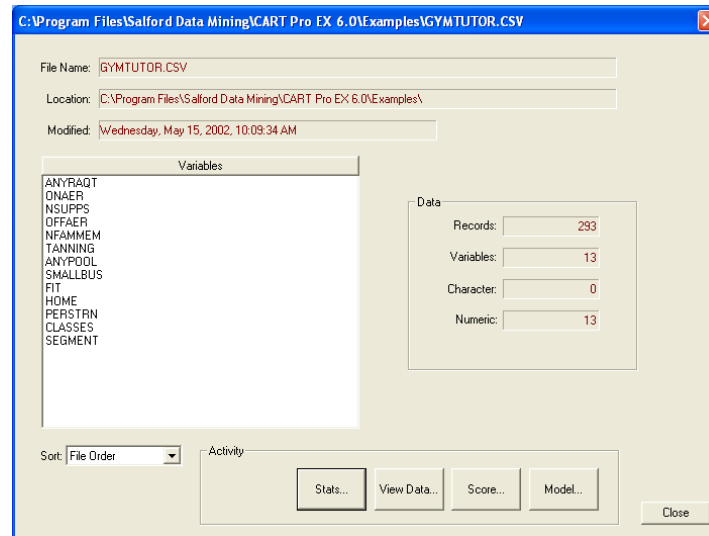
Working with Navigators

The basics of working with navigators are described in detail in Chapter 3: CART BASICS in the section titled "Tree Navigator." If you have not already read Chapter 3: CART BASICS, we encourage you to do so. It contains important and pertinent information on the use of CART result menus and dialogs.

In the next section of this chapter, we complete our exposition of the Navigator by explaining the remaining functions.

Viewing Auxiliary Variables Information

Earlier in Chapter 3: CART BASICS we set up a model based on the GOODBAD.CSV data file. Here, we set up a new but similar modeling run using GYMTUTOR.CSV with the following variable and tree type designations.



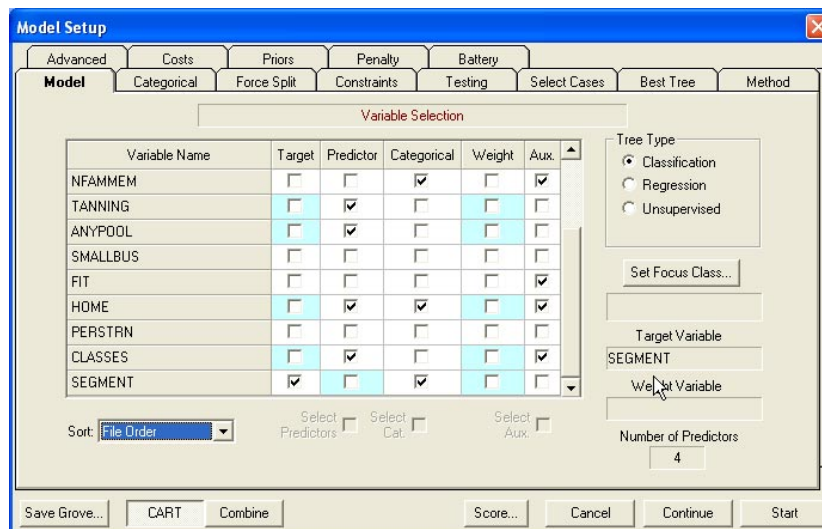
Target Variable: SEGMENT

Predictor Variables: TANNING, ANYPOOL, HOME, CLASSES

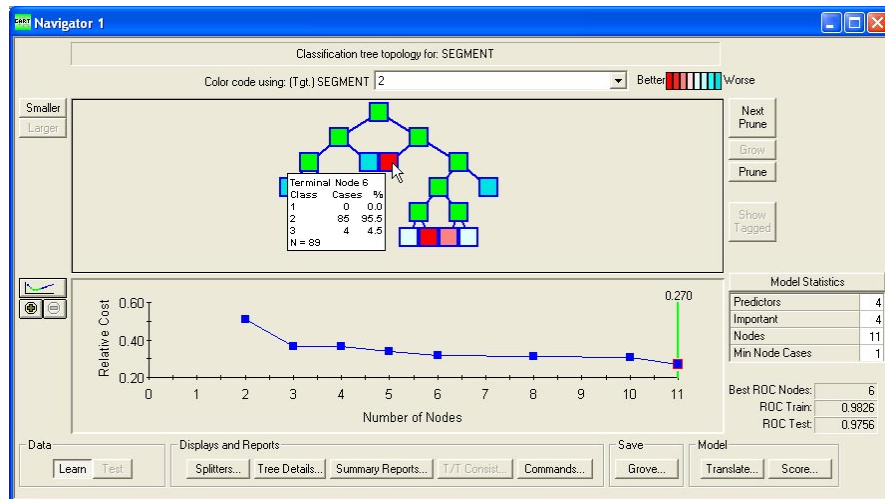
Categorical Variables: SEGMENT, HOME, NFAMMEM

Auxiliary Variables: HOME, CLASSES, FIT, NFAMMEM

Tree Type: Classification



After specifying our modeling and auxiliary variables, **[Start]** is pressed; the resulting Navigator looks as follows (color coding has been activated for SEGMENT=2):



According to the current color coding, terminal node 6 captures the majority of the second segment. Now right-mouse click on this node and choose **Auxiliary Variables**.

Navigator 2 : Terminal Node 6 Auxiliary Variables

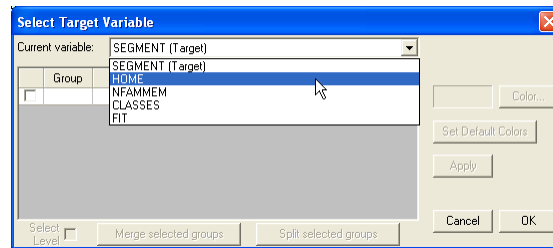
Categorical Variables	/	Value	Frequency	Pct	Cum Pct			
HOME	1		1.00000	1.12	100.00			
	0		88.00000	98.88	98.88			
Continuous Variables	∇	N	Min	Max	Mean	Missing	Sum	SD
FIT		89.00000	6.90775	10.12663	9.04828	0.00000	805.29667	0.66821
CLASSES		89.00000	1.00000	3.00000	1.15730	0.00000	103.00000	0.39597

This table reports summary statistics for HOME, CLASSES, and FIT for the given node. Frequency distributions are reported when a predictor is categorical (for example, all but one case have HOME=0), and means and standard deviations are reported for continuous predictors.

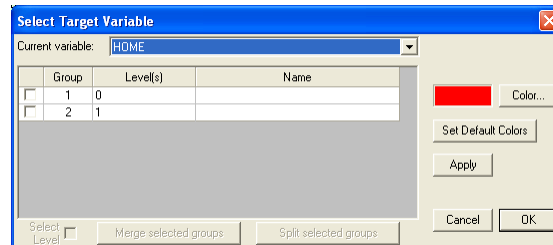
In addition to viewing the summary statistics, you may color code all terminal nodes based on any of the auxiliary variables.

For example, do the following steps to color code terminal nodes using the HOME variable:

1. Right-mouse click anywhere in the gray area in the top half of the navigator window and choose **Select Current Target...** (alternatively, use the **View->Select Current Target** menu). The Select Target Variable window will appear.

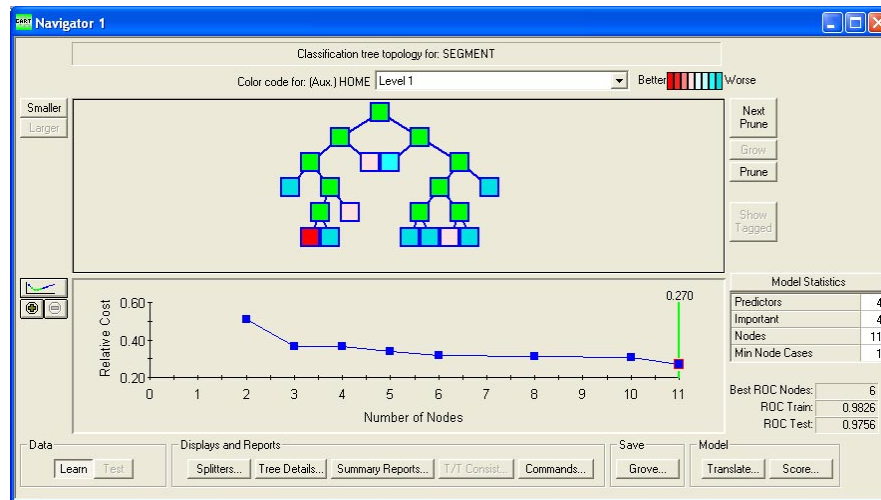


Choose HOME in the Current Variable text selection box:



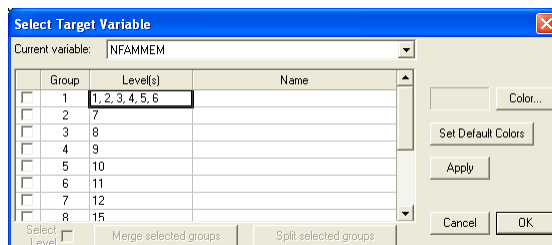
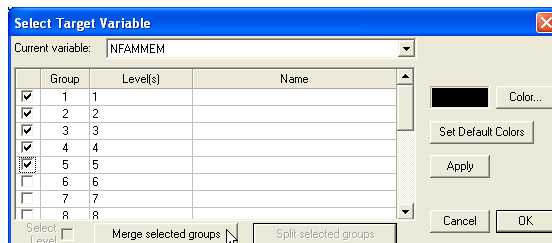
Click **[OK]**

Back in the Navigator window, choose the desired class level; the terminal nodes will now be color coded as if HOME were the target.

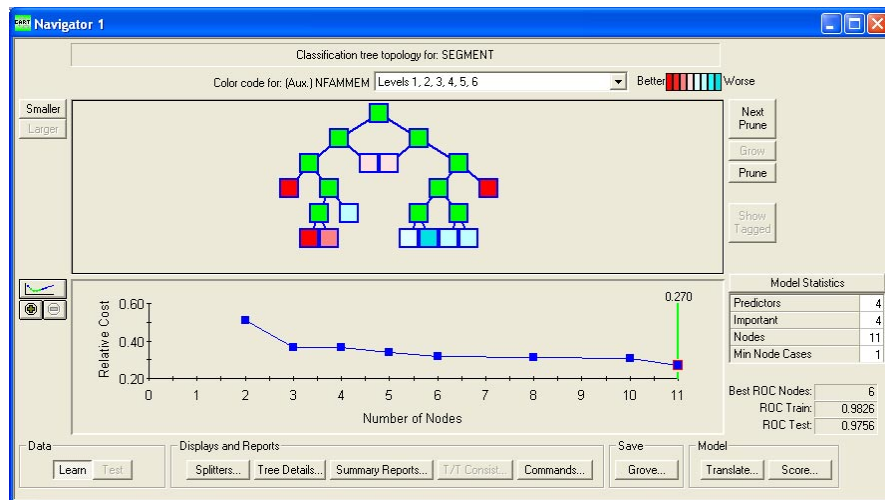


When a categorical variable has more than two levels, it is possible to group several levels to report frequency distributions for the entire group. For example, choose the NFAMMEM variable in the **Current Variable** selection box in the **Select Target Variable** window (see the steps above explaining how to get to this window).

Now put checkmarks against levels 1,2,3,4,5 and click the **[Merge selected groups]** button. As a result, all five levels are now combined into one group.



Now go back into the Navigator where you may color code terminal nodes by the group.



Similarly, you may color code terminal nodes by a continuous auxiliary variable. In this case, the color codes will be based on the mean instead of the level in focus (similar to target color coding in regression trees; see Chapter 5, Regression Trees).

- ✎ You may break the group down into original levels by checking the grouping and pressing the **[Split selected groups]** button.
- ✎ Return to the Select Target Variable dialog to return display details back to the original target variable SEGMENT.

Comparing Children

It is possible to compare two children of any internal node side by side. Simply point the mouse to the internal node, right-click, and choose the **Compare Children** menu item. A window similar to the Tree Details window shows two children side by side.

Terminal Node 1			Node 4		
TANNING <= 0.50			TANNING > 0.50		
Class	Cases	%	Class	Cases	%
1	5	100.0	1	36	33.6
2	0	0.0	2	0	0.0
3	0	0.0	3	71	66.4
W = 5.00			W = 107.00		
N = 5			N = 107		

- ✎ You can control what is reported using the **View->Node Detail...** menu just as you do for the Tree Details window.

Comparing Learn and Test

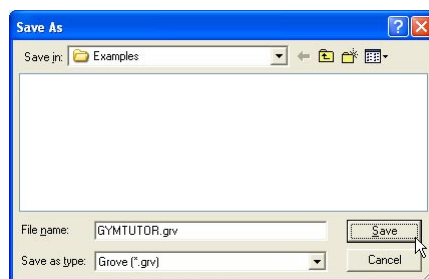
It is possible to compare learn and test node counts and percentages. Simply point the mouse to the node of interest, right-click, and choose the **Compare Learn/Test** menu item. The resulting window displays the learn and test counts and percentages by each target class.

Class	Learn N	Learn %
1	41	36.61
2	0	0.00
3	71	63.39
Total:	112	100.00

- ✎ When cross-validation trees or exploratory trees are used, only the learn counts are available, for obvious reasons.

Saving Navigator Files

CART allows you to save the Navigator to a file and then later reload it. To save a Navigator file (also known as the Grove), bring the Navigator window to the foreground and select **Save->Save Grove...** from the **File** menu. In the **Save As** dialog box, click on the **File name** text box to change the default file name. The file extension is **.GRV** and should not be changed. Select the directory in which the Navigator file should be saved and click on **[Save]**.



To open a Navigator file you have previously saved, select **Open->Open Grove...** from the **File** menu. In the **Open Grove File** dialog box, specify the name and directory location of the navigator file and click on **[Open]**.

✎ CART 6 is backwards compatible with the previous navigator file formats (*.nav, *.nv2, *.nv3). However, opening older versions will result in some new navigator features being disabled.

To open navigators from previous versions select **Open->Open Navigator...** from the **File** menu. In the **Open Tree Navigator** dialog box, specify the name and directory location of the navigator file and click on **[Open]**.

Opening a navigator in subsequent sessions allows you to continue your exploration of detailed and summary reports for each of the trees in the nested sequence or to use the navigator for scoring or translation (see Chapter 7: Scoring and Translating); however, reopening the file does not reload the model setup specifications in the GUI dialogs. To do this, you should learn the basics of command-line use in Chapter 13.

To save your model setup specifications, save the settings in a command file prior to exiting CART. The commands, by default stored in CART's command log, can be accessed by selecting **Open Command Log...** from the **View** menu (or by clicking the **Command Log** toolbar icon). To save the command log, select **Save** from the **File** menu. To then reload your setting in the Model Setup dialog, simply submit the command log. The last set of model setup commands in the command file appears in the tabbed Model.

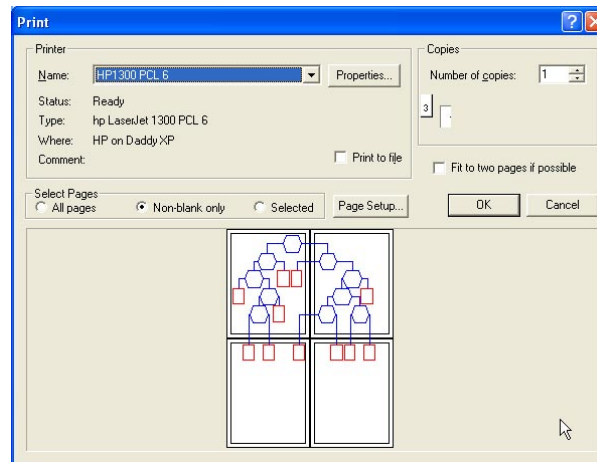


Command-line users will use the following command syntax to save CART models and navigators.

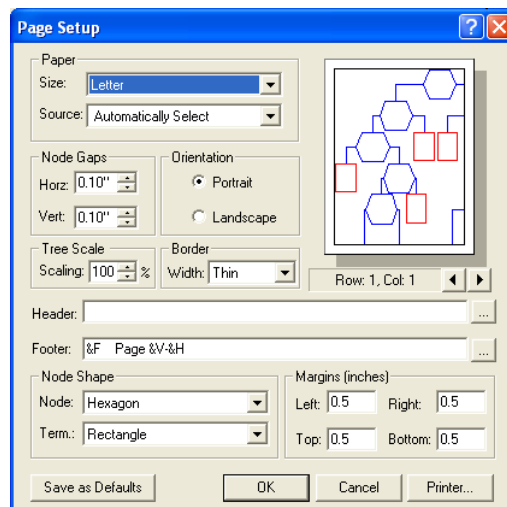
```
GROVE "<file_name.grv>"
```

Printing Trees

To print the Main Tree (or a sub-tree), bring the tree window to the foreground (click **[Tree Details...]** on the Navigator dialog) and then select **Print...** from the **File** menu (or use **<Ctrl+P>**). In the **Print** dialog box, you can select the pages that will be printed and the number of copies, as well as specify various printer properties. The Print dialog also displays a preview of the page layout; CART automatically shifts the positions of the nodes so they are not split by page breaks.



To alter the tree layout prior to printing, click the **[Page Setup...]** button. As shown below, the current layout is depicted in the tree preview window of the **Page Setup** dialog; as you change the settings, the print-preview image changes accordingly. You can use the left and right arrows just below the sample page image to change which page is previewed.



The page setup options and their default settings are:

Node Gaps [0.10"]	Change the distance between the nodes by increasing or decreasing the horizontal setting and change the height of the tree branches by increasing or decreasing the vertical setting.
Orientation [portrait]	Choose portrait or landscape.
Tree Scale [100%]	Increase/decrease the overall size of the tree.
Border [thin]	Change the width of the page border or select "no border."
Header	Enter text for header or select from the predefined settings by clicking on [...]; predefined settings include file name, tree name, column #, row #, current date and time; also included here are the alignment options (left, right, center). (Note: To include an ampersand in the header, type two ampersands, &&.)
Footer	Replace default footer text (input file name, page row and column) by entering new text or select from the predefined settings by clicking on [...]; predefined settings are similar to those for headers (see above).
Node Shapes	Change the non-terminal (node) and terminal node (term) default hexagon and rectangle shapes by clicking the down arrow and selecting an alternative shape.
Margins [0.50"]	Change left, right, top and bottom margins.

Overlaying and Printing Gains Charts

You can overlay gains charts for nested trees in a CART sequence, for different CART analyses, and for different classes of a target variable. To overlay two or more gains charts:

1. Select the corresponding navigator.
2. Click **[Summary Reports...]** and make sure the Gains Chart tab is active.

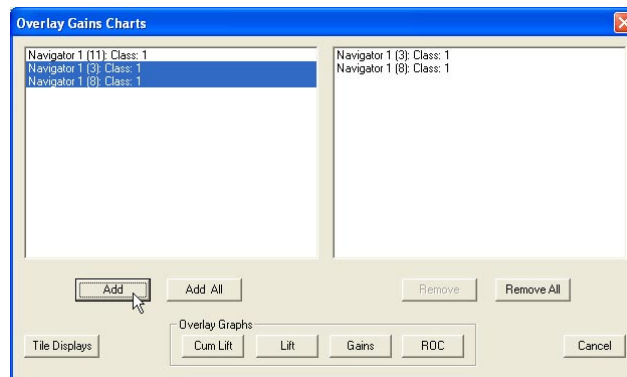


Each click on the **[Summary Reports...]** button creates a new instance of the Summary Reports window.

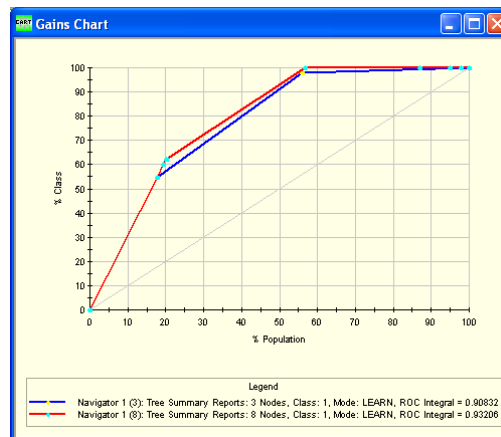
Choose the right target class in the **Tgt. Class** selection box.

Repeat steps 1 through 3 as many times as needed to have all the gains charts you would like to overlay.

Select **Gains Charts...** from the **View** menu, which will open the **Overlay Gains Charts** dialog listing the charts you want to overlay in the right panel.



Click **[Cum Lift]**, **[Lift]**, **[Gains]**, or **[ROC]** to request the corresponding overlay charts.



Each chart is displayed in a unique color with a different plotting symbol, as seen in the illustration above.

To print the contents of the Overlay Gains Chart dialog box, select **P**rint... from the **F**ile menu. To alter the layout prior to printing, select **P**age **S**etup... from the **F**ile menu.

✂ The tables in the Gains Chart, Misclassification and Prediction Success dialog boxes can also be copied and pasted into spreadsheet and word processing programs such as Excel and Word.

All of these tables and graphs can also be exported into various graphical formats. They include *.bmp, *.emf, *.jpg, *.png, and *.wmf. To export, right-click on the table or graph and select **E**xport... from the menu.



Regression Trees

*This chapter provides instructions for the steps
required to grow regression trees.*

Building Regression Trees

Our examples so far have focused on classification trees, where the target is categorical. Using regression trees, CART can also be used to analyze and predict continuous target variables. Most CART functions are shared by both classification and regression trees, but there are several important differences when we grow regression trees; these are the focus of this chapter.

Specifying a Regression Model

We develop a regression tree using the Boston Housing Price dataset that reports the median value of owner-occupied homes in about 500 U.S. census tracts in the Boston area, together with several variables that might help to explain the variation in median value across tracts. For ease of reference, definitions of the variables in BOSTON.CSV data (included with your installation sample data) are given below.

CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq. ft.
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	nitric oxides concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centers
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per \$10,000
PT	pupil-teacher ratio by town
LSTAT	% population of lower status
MV	Median value of owner-occupied homes in \$1000's

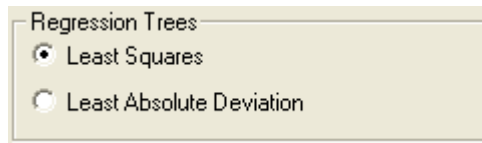
After you open a data set, setting up a CART regression analysis entails several logical steps, all carried out in one of the Model Setup dialog tabs available after clicking on the **[Model...]** button in the **Activity Window**.

Model	selects target and predictor variables, specifies categorical predictors and weight variables, chooses tree type (regression), specifies auxiliary variables
Categorical	sets up categorical class names
Force Split	specifies splitter for root node and its children
Constraints	specifies structural constraints on a tree
Testing	selects a testing or self-validation method
Select Cases	selects a subset of original data
Best Tree	defines the best tree-selection method
Method	selects a splitting rule

Penalty	sets penalties on variables, missing values, and high-level categorical predictors
Advanced	specifies other model-building options
Battery	specifies batteries of automated runs

The key differences regression tree models impose on both model setup and resulting output are:

- ◆ Certain Model Setup dialog tabs are grayed when you select the regression tree type in the Model dialog. These include the Costs and Priors tabs that provide powerful means of control over classification trees.
- ◆ Least Squares (default setting) and Least Absolute Deviation are the only splitting rules available.



✎ Even though classification splitting rules are not grayed out, the actual setting is ignored in all regression runs.


- ◆ Gains charts, misclassification tables and prediction success tables are no longer displayed in the Tree Summary Reports because they are not applicable.
- ◆ The Mean (or within-node average) of the target variable is reported for each node (rather than a class assignment) and node distributions are displayed as box plots (rather than as bar/pie graphs).

The only **required** step for growing a regression tree is to specify a target variable and a tree type in the **Model Setup—Model** tab.

If the other Model Setup dialog tabs are left unchanged, the following defaults are used:

- ◆ All remaining variables in the data set other than the target will be used as predictors (the Model tab)
- ◆ No weights will be applied (the Model tab)
- ◆ 10-fold cross validation will be used for testing (the Testing tab)
- ◆ The minimum cost tree will become the best tree (the Best Tree tab)
- ◆ Only five surrogates will be tracked and will all count equally in the variable importance formula (the Best Tree tab)
- ◆ The least squares splitting criterion for regression trees will be used (the Method tab)

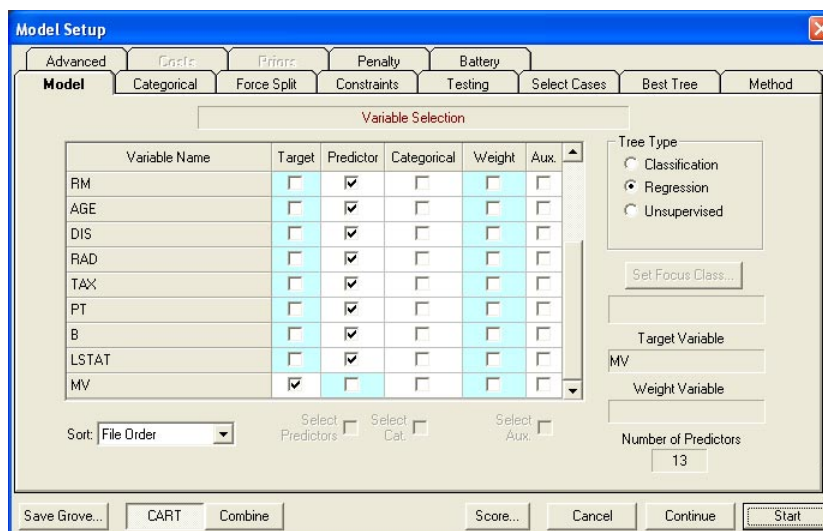
- ◆ No penalties will be applied (the Penalty tab)
- ◆ Parent node requirements will be set to 10 and child node requirements set to 1 (the Advanced tab)
- ◆ Allowed sample size will be set to the currently-open data set size (the Advanced tab)
- ◆ The 3000 limit warning for cross validation will be activated

 With respect to the command line, CART determines which tree to grow (classification or regression) depending on whether the target appears in the CATEGORY command. A classification tree is built for categorical targets and a regression tree for continuous targets.

To illustrate the regression tree concept, we use the following steps to start the analysis:

3. Select **File->Open->Data File** to open the BOSTON.CSV dataset (506 observations).

In the **Model Setup** dialog, check MV as the target variable and click on the **Regression Tree** radio button. Check all the other variables as predictors.



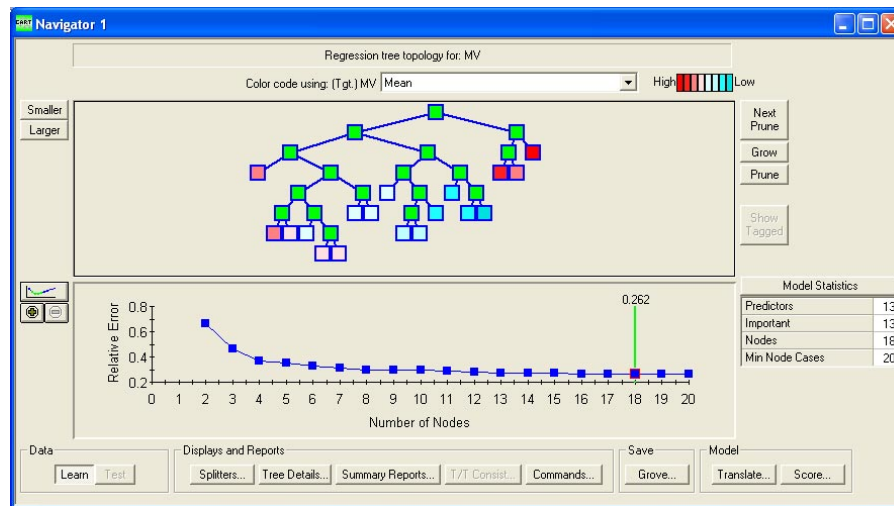
In the **Model Setup—Advanced** tab, set “Parent Node Minimum Cases” to 40 and “Terminal Node Minimum Cases” to 20. This will ensure that the terminal nodes will not become too small.

Minimum Node Sizes		
	Unweighted	Weighted
Parent node minimum cases:	40	
Terminal node minimum cases:	20	

Click **[Start]**.


Tree Navigator

At the end of the model-building process, a navigator window for a regression tree will appear.

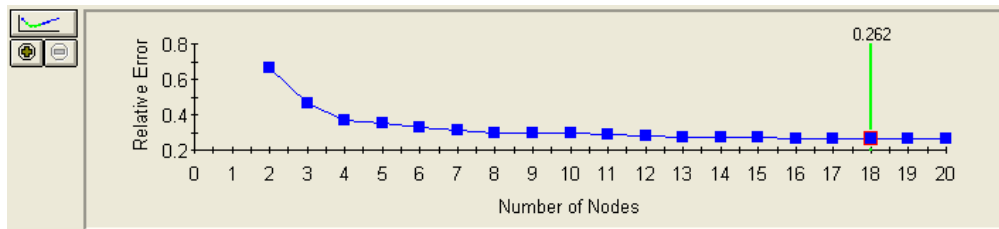


By default, CART uses the least squares splitting rule to grow the maximal tree and cross-validated error rates to select the “optimal” tree. In this example, the optimal tree is the tree with 18 terminal nodes, as displayed in the Navigator above.

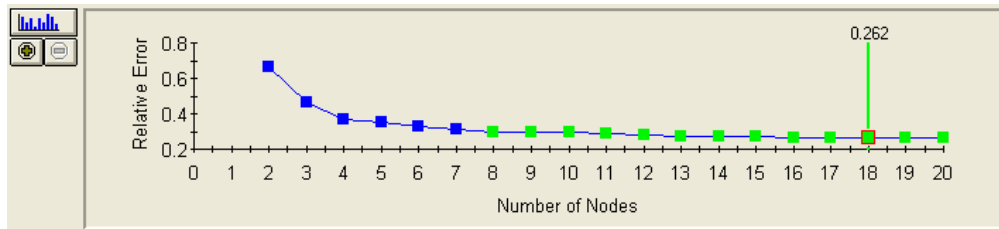


The upper button in the  group cycles over three possible display modes in the lower part of the Navigator Window:

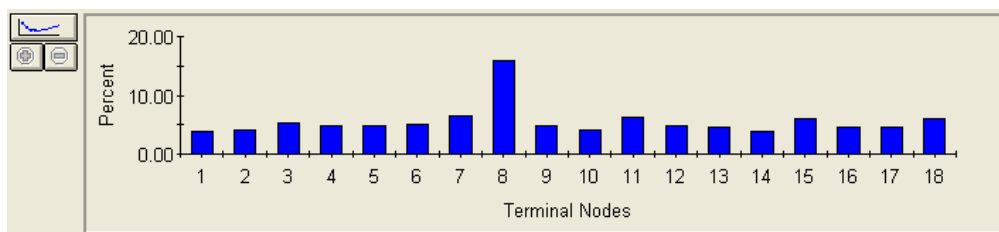
Default Mode shows the relative error profile (either Test, Cross-Validated, or Learn depending on the testing method chosen in the Testing tab of the Model Setup window):



1-SE Mode shows the relative error profile where all trees with performance within one standard error of the minimal error tree are marked in green:



Node Size mode shows the node size bar chart for the currently-selected tree:



You can click on any of the bars to see the corresponding node highlighted in yellow on the tree display.

To change the currently-selected tree, go to one of the previous modes, pick a new tree, and switch back to the Node Size mode.

- ✎ The tree picture can be made smaller or larger by pressing the corresponding buttons in the left upper corner of the navigator window.
- ✎ As with classification trees, to change the level of detail you see when hovering over nodes, right-click on the background of the Navigator window and select your preferred display from the local pop-up menu.

Copy
Add to Report
Export...
Select Current Target...
VARIABLE
VARIABLE <= 45
VARIABLE <= 45
Q1 42.1
Median 85.6
Q3 98.1
N = 1040
VARIABLE <= 45
Min 14
Q1 42.1
Mean 84.1
Median 85.6
Q3 98.1
Max 99.4
N = 1040

✎ The **[Learn]** and **[Test]** group of buttons controls whether Learn or Test data partitions are used to display the node details on the hover displays or all related Tree Details windows.

Color Coding

The terminal nodes can be color coded by either target mean or median. Make your selection in the **Color Code Using:** selection box.

Viewing Tree Splitters and Details

The **[Splitters...]** button and the **[Tree Details...]** buttons work similarly to the classification case described previously (see Chapter 3: CART BASICS). The only difference is that node information now displays target means and variances instead of frequency tables and class assignments.

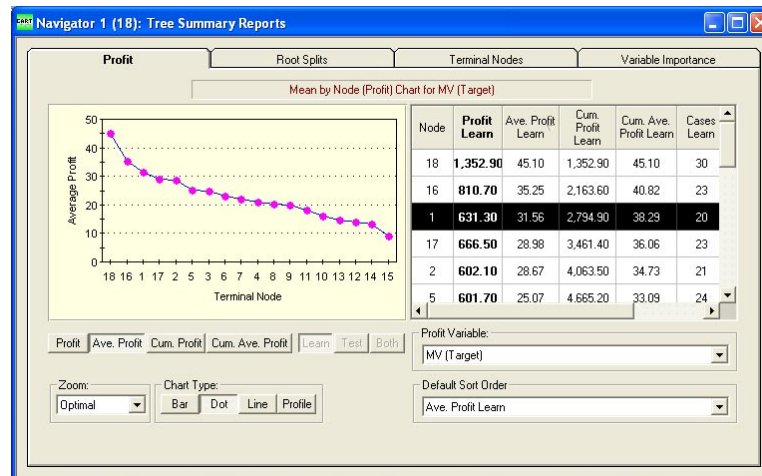
The Tree Details display can be configured using the **View—Node Detail...** menu.

Regression Tree Summary Reports

The overall performance of the current tree is summarized in the four Summary Reports dialog tabs. To access the reports, click the **[Summary Reports...]** button at the bottom of the Navigator window (or select **Tree Summary Reports...** from the **Tree** menu).



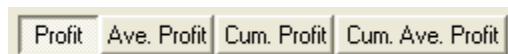
The **Profit** tab provides a useful model summary in terms of the profit associated with each node. It is assumed that each record in a dataset is associated with a certain continuous amount of profit. This information is either represented by the continuous target itself (in which case the profit value is the actual target of modeling), or by any other continuous variable present in the dataset (cross-evaluation of model).



First, choose the **Profit Variable** carrying information about the profit associated with each record in the dataset. By default, this variable is set to the target variable in regression runs; however, it could be changed to any of the continuous auxiliary variables that were specified in the Model tab of the Model Setup dialog.

Second, specify the **Default Sort Order**. This setting will control how the terminal nodes of the currently-selected tree are ordered on the table and the graph above. Currently, sorting either by **Profit Learn** (node sum of profit values in the Learn data) or **Average Profit Learn** (Profit Learn divided by node size) is available.

Third, choose one of the four possible measures to be displayed on the vertical axis of the graph by pressing the following group of buttons:



Profit—within-node accumulated profit.

Ave. Profit—Profit divided by the node case count.

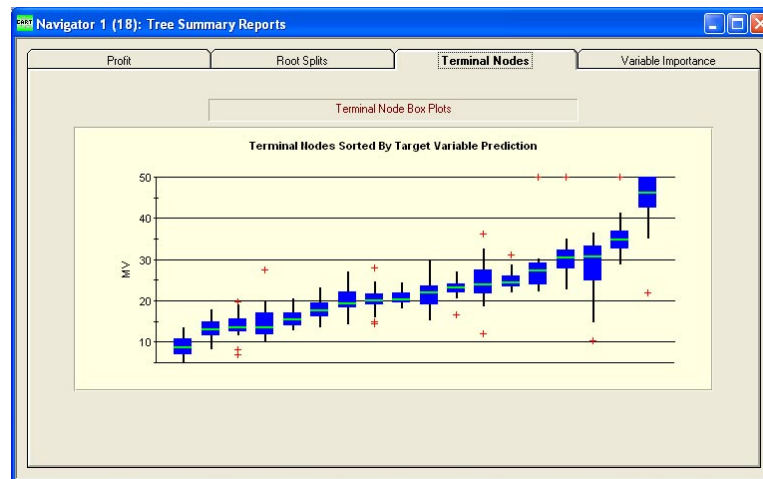
Cum. Profit—same as Profit but accumulated over all nodes in the sorted sequence up until the current node.

Cum. Ave. Profit—Cum. Profit divided by the total number of cases in all nodes in the sorted sequence up until the current node.

- ✂ All four measures, as well as node case counts, are reported on the table.
- ✂ In the presence of the explicit Test sample, the user can also choose among Learn, Test, and Pooled measures using the corresponding buttons.
- ✂ The Zoom and Chart Type controls change the visual appearance of the graph.

Terminal Nodes

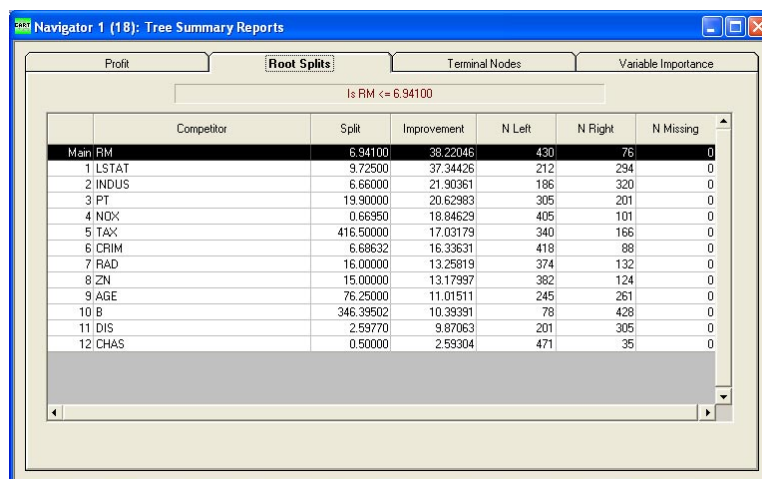
The **Terminal Nodes** tab displays box plots for the node distributions of the target sorted by the mean. Hover over any of the boxes to see detailed information about the node.



- ✂ When separate learn and test parts of the data are used, **[Learn]** and **[Test]** buttons allow switching between learn and test distributions. No matter which button is pressed, the nodes are always sorted by the learn means to quickly assess node stability.

Root Splits

The **Root Splits** lists ALL root node competitors sorted in descending node by split improvement. The report also shows split details in terms of case counts.



Competitor	Split	Improvement	N Left	N Right	N Missing
Main RM	6.94100	38.22046	430	76	0
1 LSTAT	9.72500	37.34426	212	294	0
2 INDUS	6.66000	21.90361	186	320	0
3 PT	19.90000	20.62983	305	201	0
4 NOX	0.68950	18.84629	405	101	0
5 TAX	416.50000	17.03179	340	166	0
6 CRIM	6.68632	16.33631	418	88	0
7 RAD	16.00000	13.25819	374	132	0
8 ZN	15.00000	13.17997	382	124	0
9 AGE	76.25000	11.01511	245	261	0
10 B	346.39502	10.39391	78	428	0
11 DIS	2.59770	9.87063	201	305	0
12 CHAS	0.50000	2.59304	471	35	0

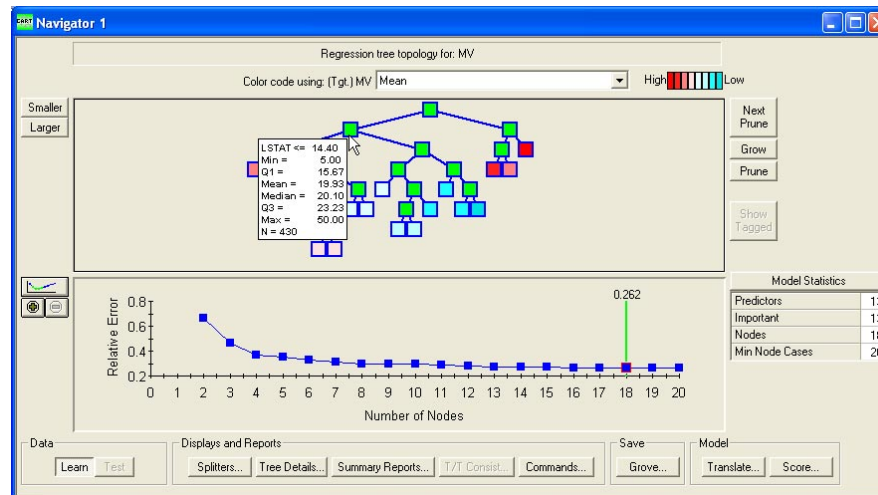
While the competitor information is also available for all internal nodes by clicking on the node itself, it is usually limited to only the top five entries.

Variable Importance

The **Variable Importance** tab: same as classification but importance scores are now based on regression improvements. (See Chapter 3: CART BASICS for discussion of Variable Importance.)

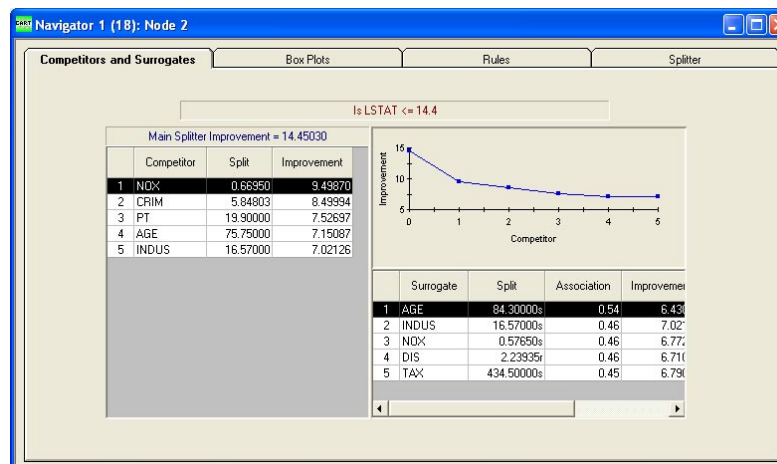
Detailed Node Reports

To see what else we can learn about our regression tree, return to the Navigator by closing the Summary Reports window. To request a detailed node information display, simply click on the node of interest; for example, left-click on the left child of the root node (internal node 2).



The Competitors and Surrogates tab

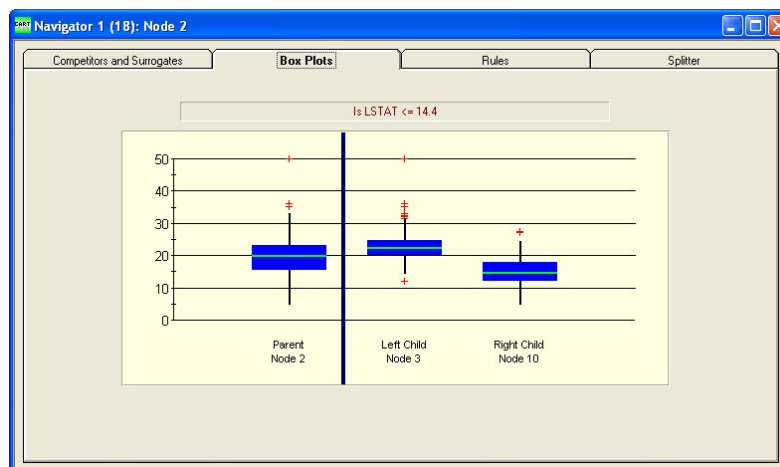
As illustrated below, the first of the four tabs in the non-terminal node report provides node-specific information on both the competitor and surrogate splits for the selected node (in this case, the root node). This results tab is discussed in detail in Chapter 3: CART BASICS.



The Box Plots tab

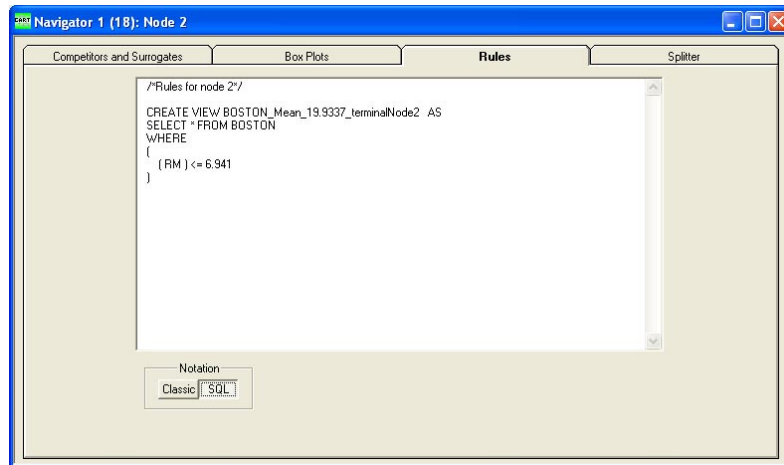
The **Box Plots** tab shows the current node box plot on the left-hand side and two children box plots on the right-hand side. This helps to interpret the nature of the split.

The blue box depicts the inter-quartile range, with the top of the box (or upper hinge) marking the 75th quartile and the bottom (lower hinge) marking the 25th quartile for the target variable MV. The horizontal green line denotes the node-specific median while the whiskers (or upper and lower fences) extend to plus/minus 1.5 times the inter-quartile range. Red plusses represent values outside the fences, usually referred to as “outliers.”



The Rules tab

The third tab in the node report, the **Rules** tab, is displayed as follows. For reference, we display the Rules tab for Node 2. Non-terminal and terminal node reports (with the exception of the root node) contain a Rules tab. This tab is discussed in detail in Chapter 3: CART BASICS.



The Splitter tab

When the main splitter is continuous, the left- and right-child summary statistics of the target are displayed in table form.

Is LSTAT <= 14.4			
	Left Target's statistics		Right Target's statistics
Min	11.90000		5.00000
Mean	23.34980		14.95600
Max	50.00000		27.50000
N	255.00000		175.00000

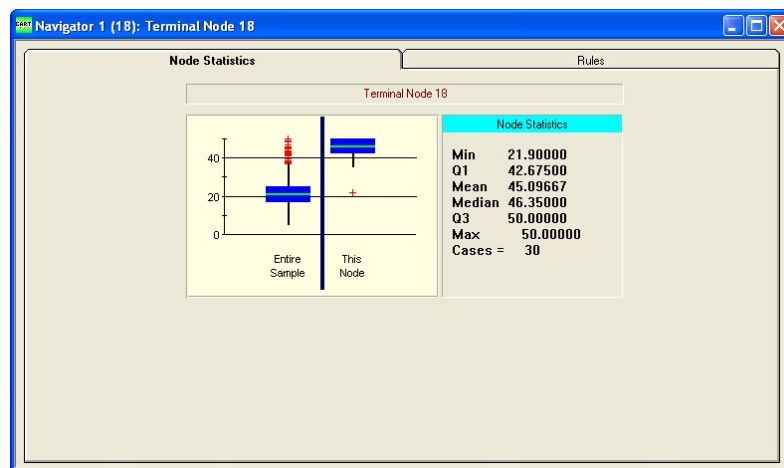
When the main splitter is categorical, the partition of the splitter's levels between the left and right sides is displayed. This results tab is discussed in more detail in Chapter 3: CART BASICS.

Terminal Node Report

To view node-specific information for a terminal (red) node, click on the terminal node (or right-click and select **Node Report**). For our example, left-click on terminal node 18 (far right terminal node).

The Node Statistics tab

The **Node Statistics** tab shows the current node target box plot in comparison with the target box plot for the root node (the entire learn sample). This helps us to see whether the high-end or the low-end segment of the population is contained in the current node. Node-specific summary statistics are also reported. Both the color-coding and the relative position of this node compared to the root node suggest that the highly-priced segment is contained in this node.



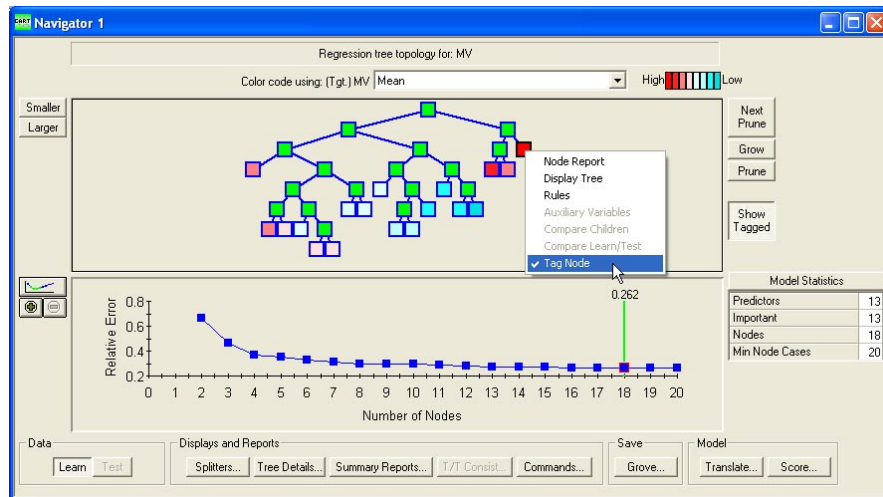
The Rules tab has been described above.

For further discussion of regression tree modeling, splitting rules, and interpreting regression node statistics, see the CART Reference Manual.

Viewing Rules

There are several flexible ways to look at the rules associated with an entire tree or some specific parts of the tree.

In the Navigator window, you can tag terminal nodes for further use by hovering the mouse over, right-mouse clicking, and selecting the Tag Node menu item. In the following example we tagged all nodes color coded in red and pink (high-end neighborhoods).

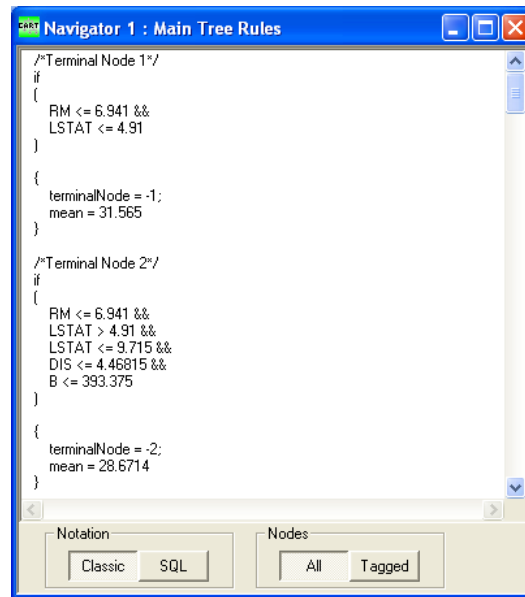


Next we request an overall Rules display either via **View->Rules...** menu or by right-mouse clicking on the root node and choosing the **Rules** item.

The resulting window contains rules for the entire tree when **[All]** is pressed or only for the tagged terminal nodes when **[Tagged]** is pressed.

Both **Classic** and **SQL** rule notations are supported.

You can also limit the rules display to a specific branch in a tree by right-mouse clicking on the branch root and choosing the **Rules** item. The resulting window will only list rules for the terminal nodes covered by the selected branch as well as rules leading to the given branch.



- ✂ The Main Tree Rules display only gives node-based rules, ignoring missing value handling mechanisms entirely.
- ✂ To request a full display of the tree logic, including missing value handling, check the chapter called **Translating Model** in this manual.

Chapter

6



Ensemble Models and Committees of Experts

The multi-tree methods:

Bootstrap Aggregation and ARCing.

Building an Ensemble of Trees

Researchers began exploring the potential value of building multiple trees around 1990. The core idea was that if one tree is good then maybe several trees would be even better. The best known of the straightforward ensembles is Leo Breiman's bagger, which is the main topic of this chapter. Subsequently, Breiman also introduced Random Forests, now available as a separate Salford Systems module.

Simple ensembles generate predictions by averaging the outputs of independently built models. The more complex method of boosting builds a sequence of trees, with each new tree attempting to repair the errors made by predecessor trees. Boosting was first introduced by Freund and Schapire (1996), who showed how a three-tree model could outperform a single tree. Later, a number of researchers explored the boosting of many trees. Leo Breiman (1996) observed that a simple modification to the bagger would yield a method very similar to boosting; that method (ARcing) is also discussed briefly in this chapter.

A newer and far more powerful form of boosting is available in the Salford Systems TreeNet module.

Bootstrap Aggregation and ARcing

In addition to growing a classification or regression tree, you may switch to either bootstrap aggregation (bagging) or adaptive resampling and combining (ARcing) mode by pressing the **[Combine]** button in the **Model Setup** dialog. The Combine tab is now available—the command center to set up various bagging and ARcing controls.

The Testing and Best Tree tabs are not available because they are used only in single tree modeling.

CART's Combine dialog allows you to choose from two methods for combining CART trees into a single predictive model. In both bootstrap aggregation (bagging) and Adaptive Resampling and Combining (ARcing), a set of trees is generated by resampling with replacement from the original training data. The trees are then combined by either averaging their outputs (for regression) or by using an unweighted plurality-voting scheme (for classification).

Bagging versus ARcing

The key difference between bagging and ARcing is the way each new resample is drawn. In bagging, each new resample is drawn in an identical way (independent samples), while in ARcing the way a new sample is drawn for the next tree depends on the performance of the prior trees.

Bootstrap Resampling

Bootstrap resampling was originally developed to help analysts determine how much their results might have changed if another random sample had been used instead, or how different the results might be when a model is applied to new data. In CART, the bootstrap is applied in a novel way - a separate analysis is conducted for each resample or replication generated, and then the results are averaged. If the separate analyses differ considerably from each other (suggesting tree instability), averaging will stabilize the results, yielding much more accurate predictions. If the separate analyses are very similar to each other, the trees exhibit stability and the averaging will neither harm nor improve the predictions. Thus, the more unstable the trees, the greater the benefits of averaging.

When training data are resampled with replacement, a new version of the data is created that is a slightly “perturbed” version of the original. Some original training cases are excluded from the new training sample, whereas other cases are included multiple times. Typically, 37 percent of the original cases are not included at all in the resample; the sample is brought up to full size by including other cases more than once. A handful of cases will be replicated 2, 3, 4, 5, 6, or even 7 times, although the most common replication counts are 0, 1 and 2. The effect of this resampling is to randomly alter the weights that cases will have in any analysis, thus shifting slightly the results obtained from tree growing or any other type of statistical analysis.

Adaptive Resampling and Combining

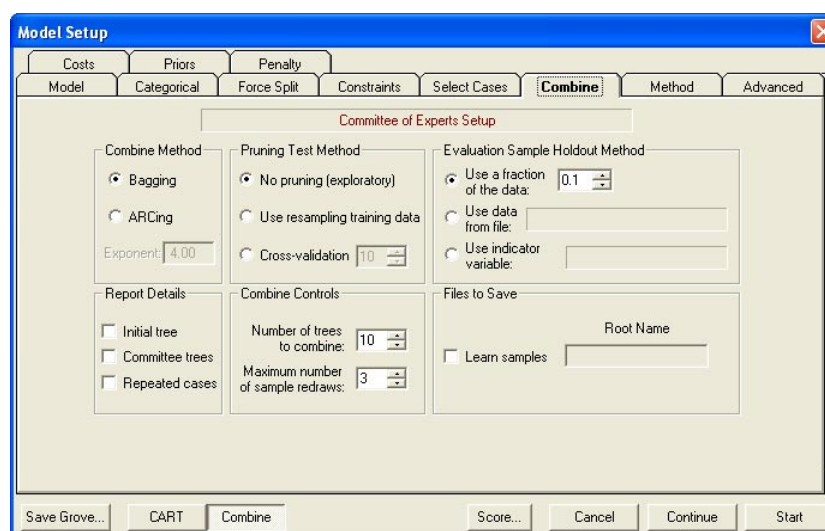
ARCing, Leo Breiman’s (1996) variant on the boosting procedure first introduced by Freund and Schapire (1996), performs as well as or better than boosting. In ARCing, the probability with which a case is selected for the next training set is not constant and is not equal for all cases in the original learn data set; instead, the probability of selection increases with the frequency with which a case has been misclassified in previous trees. Cases that are difficult to classify receive an increasing probability of selection while cases that are classified correctly receive declining weights from resample to resample. Note, however, that as the probability of selection becomes more skewed in favor of the difficult-to-classify cases, the probability of selection for the typical case quickly declines to zero and the process of sample building takes an increasingly longer time.

In general, we recommend bagging rather than ARCing because bagging is more robust with dependent variable errors and also much faster. Nevertheless, ARCing is capable of remarkably reducing predictive error. Note also that both bagging and ARCing generate a “committee of experts” rather than a single “optimal” tree. Because a single tree is not displayed, no simple way exists to explain the underlying rationale driving the averaged predictions. In this sense, combined trees are somewhat akin to the black box of a neural net, although the trees are built much more quickly.

- One final caution on combining via bagging or ARCing: the increase in accuracy is sometimes accomplished for the class in which you have the least interest. For example, in a binary response model in which response is relatively rare, bagging and ARCing may improve the non-response classification accuracy while slightly reducing the response classification accuracy relative to a standard CART tree. We recommend that you experiment with adjusting the priors setting to induce the most useful improvements.

The Combine Tab

The **Model Setup—Combine** tab allows you to specify various advanced committee tree-building control options and settings.



Combine Method

The **Combine** dialog houses the command controls for both bagging and ARCing. To build a committee of experts tree, first select either Bagging or ARCing. If you select ARCing, you will need to specify the exponent or power setting as well. Power sets the weight the resampling puts on selecting cases that have been previously misclassified; the higher the power, the greater the bias against selecting cases that were previously classified correctly. Breiman has found that a power setting of four works well, while settings of one or two give results virtually identical to bagging.

Setting the power greater than four could make it difficult to locate a sample large enough to fill the training sample if only a small fraction of the data is misclassified. Also, as Dietterich (1998) has reported, if the dependent variable is recorded in error,

then using ARChing will progressively focus new trees on the bad data, yielding poor predictive models.

Combine Controls

After selecting bagging or ARChing, the next step is to select the number of trees you want to grow. Bagging typically shows good results with about 100 trees, but ARChing may require up to 250 trees. The number of trees is initially set at 10 and can be changed by entering a new value in **Number of Trees to Combine**. We recommend you first experiment with a modest number to see how the procedure is working. If it looks promising, launch a CART run with a full complement of 100 or more trees.

As noted above, when using ARChing, as the probability of selection becomes more skewed in favor of difficult-to-classify cases, the probability of selecting the typical case quickly declines to zero and the time for sample building increases. In many runs, the ARC process of resampling will simply bog down and the ARCher will automatically reset the probabilities to their equal starting values and continue generating additional trees.

The option **Maximum Number of Sample Redraws** enables you to control how hard the ARCher should try to build a sample. The default setting is three. If CART cannot build one of the trees in the resampled series, you can increase the maximum number of redraws and try again.

Pruning Test Method

When growing a single tree, pruning is not merely optional; it is vital to obtaining a reliable tree. By definition, a CART tree is first overgrown (i.e., overfit) and then the overfit portions are pruned away with the help of a test or cross validation data set.

When combining trees, Breiman has shown that the trees need NOT be pruned because whatever overfitting may result is averaged away when the combining takes place. For this reason No Pruning is the default setting when using either bagging or ARChing.

✂ The other two pruning methods are available for historical reasons only.

Evaluation Sample Holdout Method

A holdout sample is used to evaluate the performance of the committee of experts tree generated via bagging or ARChing. The holdout sample is NOT used to build or prune any tree, but rather is used only to evaluate the predictive capability of both the committee of experts tree and the initial tree built on the full sample.

The three options for specifying the holdout data set are grouped in the Evaluation Sample Holdout Method box:

1. Use a fraction of the data (specify fraction; default=0.10).
2. Use a separate data set (select data set).
3. Use an indicator variable (select name of test binary dummy variable).

Files to Save

To save individual learn samples (obtained using sampling with replacement) simply checkmark the **Learn samples** box and specify the **Root Name**, say “learn.”

Because CART will attach a serial number to the root names of the learn files, we recommend keeping the names to six characters or less to avoid truncation. The serial number corresponds to the resample cycle number (e.g., if cycles=10, the learn samples will be labeled learn01, learn02 ... learn10).

To grow the committee of experts, click **[Start]**.

The combine model can be saved into a grove file for further scoring or translating by pressing the **[Save Grove...]** button and specifying the file name before the model is built.

✎ The grove file in this case will have multiple trees and does not have an accompanying navigator file.

Report Details

By default, the Combine text output consists of summary statistics for the train (learn) sample and the holdout sample as well as a prediction-success (or confusion matrix) report summarizing how well the holdout sample performed on the initial tree (built using the in- and out-of-bag data) relative to the committee of experts tree. The prediction-success tables for the committee and for the initial tree are also displayed in the Combine Results dialog (see example below).

Committee: 1

Committee Tree Initial Tree

Committee Prediction Success

Actual Class	Total Cases	Percent Correct	Predicted Class		
			1 N=6	2 N=7	3 N=11
1	6	100.00	6	0	0
2	9	66.67	0	6	3
3	9	88.89	0	1	8
Total:			24.00		
Average:			86.15		
Overall % Correct:			83.33		

Count Row % Column %

Grove: C:\Salford\TestTemp\s1fs51 Data: C:\Program Files\...\GYMTUTOR.CSV

Save Grove...

In the Report Details group box you can change the default report contents as well as request the following additional reports:

- ◆ Initial tree - standard text report (tree sequence, node details, etc.) for the tree grown on the entire in- and out-of-bag data
- ◆ Committee trees - standard text report for each “expert” tree grown in the series
- ◆ Repeated cases - summary tables displaying the proportion of observations repeated in each resample (displayed for each committee tree and for the committee as a whole)

Given that the initial tree is constructed using CART’s default tree-building settings, another benchmark you may want to consider when evaluating the performance of your committee of experts is a single CART tree built using options appropriate for your particular application (e.g., you may want to experiment with different splitting rules, priors, costs, etc.).



Scoring and Translating

This chapter provides instructions for the steps required to internally and externally apply models to new data.

Scoring and Translating Models

No predictive modeling process would be complete without the ability to apply your models to new data. CART 6 offers two ways to do this, internally by using CART's built-in scoring facility or externally by first translating your models into any of the supported languages (SAS[®] compatible, C, or PMML).

This section describes how to use the internal SCORE command to predict a target variable using either new data or old (learn) data. The process of using a CART tree to predict a target variable is known as “dropping data down a tree” or “scoring” data. Each observation is processed case-by-case, beginning at the root node. The splitting criteria are applied, and in response to each yes/no question, the case moves left or right down the tree until it reaches a terminal node. If the primary split criterion cannot be applied because the case is missing data, a surrogate split criterion is used. If no surrogates are available, the case moves down the tree with the priors-adjusted majority.

- ✂ In CART 6, unlike previous versions of CART, you may score any tree from the pruning sequence without any extra steps involved.
- ✂ Because of the new mechanism, the SELECT command and the **Select Tree** menu item are no longer supported. To obtain classic output on a tree other than the optimal, you should translate that tree into LANGUAGE=CLASSIC (see the Translating section below).

Navigator Files versus Grove Files

Previous versions of CART produced both navigator and grove files. CART 6 combines the two types of information and stores it all in a grove file.

A grove file is a binary file that stores all the information about the tree sequence needed to apply any tree from the sequence to new data or to translate the tree into a different presentation. Grove files contain a variety of information, including node information, the optimal tree indicator, and predicted probabilities. Grove files are not limited to storing only one tree sequence, but may contain entire collections of trees obtained as a result of bagging, arcing, or cross validation. The file format is flexible enough to easily accommodate further extensions and exotic tree-related objects such as TreeNet models.

Navigator files, on the other hand, serve the sole purpose of presenting a single-tree sequence using the GUI back end, also known as the Navigator window. In the previous chapters, many examples of using navigator displays to analyze trees and present the results were provided.

To save a CART user the trouble of keeping track of two different files, CART 6 embeds a corresponding navigator file into the grove file whenever the latter is saved (unless the user explicitly turns off this feature).

Here we provide a brief reminder of the multiple ways to save a grove file:

1. When the Navigator window is active, you may save the corresponding navigator and grove files by clicking the **[Grove...]** button.
2. Issuing the GROVE “<file_name.grv>” command results in a copy of the grove file that will be embedded in the navigator.



The GROVE command names a grove file in which to store the next tree (or committee or group of trees). Its syntax is:

```
GROVE <filename>
```

When a grove file is embedded into a navigator file, you may easily save it separately by first opening the navigator file in the GUI (**File->Open->Open Navigator**) and then pressing the **[Save Grove]** button.

For example, let's make a default CART run for the GYMTUTOR.CSV data. To begin, simply mark SEGMENT as the target and press **[Start]**. When complete with the Navigator in the foreground, press **[Grove...]**. In the resulting **Save As** dialog, choose the name of the file and the folder to which you want the file saved. Finally, press the **[Save]** button. The grove file (extension *.grv) is now saved. Furthermore, it has the navigator embedded in it.

You now have all you need to proceed with scoring or translating.



Alternatively, you may request that grove and navigator files be saved as part of the model-building process. Simply press the **[Save Grove...]** button in the **Model Setup** window and enter the file name and directory. When the model is finished, both the grove file and the embedded navigator will be saved.

The above procedure is equivalent to placing the GROVE “<file_name.grv>” command before the BUILD command in your command file.



The default target folder for the grove files can be set in the **Output Files>Model Information** section of the **Options-Directories** tab when selecting the **Edit->Options** menu.

Converting a Tree File to a Grove File

The earliest versions of CART stored model information in a tree file (extension *.tr1). Tree files had the severe limitation of containing information on a single tree only (usually the optimal tree). For backward compatibility we have added a command that allows you to translate any *.tr1 file into a grove file. Of course, the resulting grove file still has only one tree.



To translate an old tree file “old_tree.tr1” into a grove file “old_tree.grv,” use the following command syntax:

```
GROVE "old_tree.grv" IMPORT="old_tree.tr1"
```

Scoring CART models

Scoring will differ depending on whether you are working with a grove file or a grove file embedded in a navigator created with CART 5 or CART 4.

Scoring Using a CART 5 Navigator with an Embedded Grove File

The navigator window must be open and active.

1. Press the **[Score...]** button.
2. Enter the relevant information into the Score Data window (described below).
3. Press **[OK]** to activate the scoring process.



The **Grove File** portion of the **Score Data** window will contain your navigator file name—this means that the embedded grove file will be used for scoring. You do not have to change this unless you want an external grove file to be used for scoring.



This mode may not be available for all older navigators.


Scoring Using Only the Grove File

If you have a grove file you would like to use for scoring, do the following steps:

1. Make sure the **CART Output** window is active.

Choose **Score Data...** in the **Model** menu.



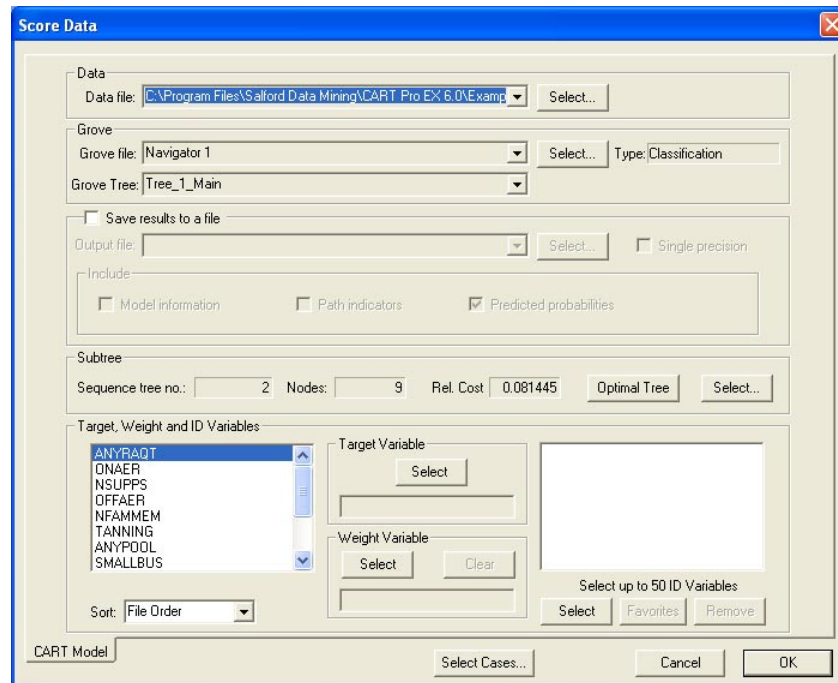
Both the above steps can be replaced by simply clicking the  button in the toolbar.

Enter relevant information into the **Score Data** dialog, including the name of the grove file in the **Grove File:** section.

Press **[OK]** to activate the scoring process.

Score Data Dialog

The **Score Data** dialog is shown in the picture below.



Data File

Click the **[Select...]** button next to this line to select the data file you want to score. By default, CART puts the most recently-opened data file into this field.

Grove File

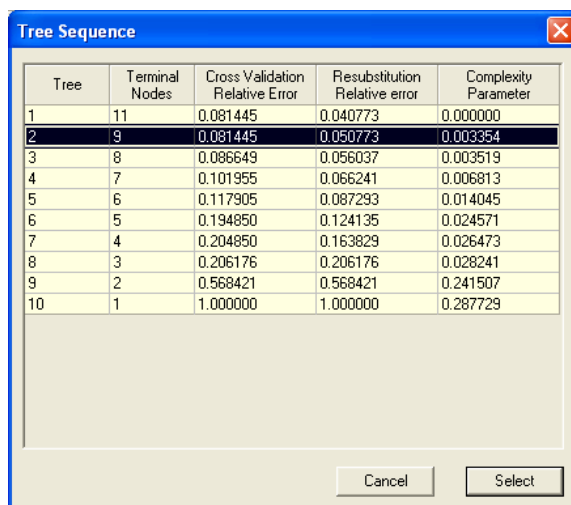
Click the **[Select...]** button to pick an external grove file for scoring or leave this field unchanged if you are scoring from a navigator file with an embedded grove file (in which case the navigator file name will appear in the field).

Save Results to a File

Place a mark in the **[x] Save results to a file** check box if you want the results of scoring saved into a separate data file. Press the **[Select...]** button to specify the output file's name, location, and format. Decide whether you want **Model Information**, **Path Indicators**, and **Predicted Probabilities** included in the output dataset (see details below).

Subtree

Click **[Select...]** if you want to score other than the optimal tree and then choose the tree in the **Tree Sequence** dialog. By default CART scores the optimal tree, which you may always return to by pressing the **[Optimal Tree]** button.



Tree	Terminal Nodes	Cross Validation Relative Error	Resubstitution Relative error	Complexity Parameter
1	11	0.081445	0.040773	0.000000
2	9	0.081445	0.050773	0.003354
3	8	0.086649	0.056037	0.003519
4	7	0.101955	0.066241	0.006813
5	6	0.117905	0.087293	0.014045
6	5	0.194850	0.124135	0.024571
7	4	0.204850	0.163829	0.026473
8	3	0.206176	0.206176	0.028241
9	2	0.568421	0.568421	0.241507
10	1	1.000000	1.000000	0.287729

Cancel Select

- ✎ The current tree number in the pruning sequence (starting with the largest tree and going backwards), the number of terminal nodes, and the relative cost are reported for your convenience.

Target, Weight, and ID Variables

If a target variable in your data set has a name other than the name used in the original learning dataset (proxy target variable), then you should highlight it in the left panel and press **[Select]** in the **Target Variable** area.

- ✎ If the target name is the same as the original, simply skip the above step—CART will detect this automatically. CART will also handle the case when there is no target at all; however, for obvious reasons, some of the scoring results reports will become unavailable.

Select the weight variable, if any, by highlighting it in the variable list panel and pressing the **[Select]** button in the **Weight Variable** area.

Finally, select up to 50 **ID variables** in the variable list panel and add these to the right panel by pressing the corresponding **[Select]** button.

- ✎ An ID variable could be any variable that was NOT part of the final model (target and finally-selected predictors). Check the **Model Information**

checkbox in the **Include** area if you want the original target and predictors propagated into the output dataset (see below).

Output Data Set

CART saves an output dataset as a result of scoring whenever **[x] Save Results to a file** is checked in the **Score Data** dialog. Depending on your settings, different variables may appear in the output dataset.

Variables that are always created

CASEID	a record number identifier.
REPONSE	a predicted response (class assignment for classification trees or node average for regression trees).
NODE	node assignment (useful when working with hybrid models).

- Depending on the file format, having an original target called **RESPONSE** and checking “Model Information” (see below) will result either in two variables with identical names (one for the predicted response and one for the actual response) or in distinguishing the original response by renaming it **RESPONSE1**. We suggest avoiding this situation to eliminate possible complications.

Variables created only when a target exists

CORRECT	binary indicator telling if the predicted response is the same as the actual response.
----------------	--

When “Model Information” is checked

CART will include the original target (if available) and all predictors that participate in the model, that is, that have non-zero variable importance scores.

When “Path Indicators” is checked

PATH_<N>	Each of these variables gives the node number that a case goes through at the <N>th depth in the tree. PATH_1 is always set to 1 to indicate that the first node is the root node. Positive numbers refer to internal nodes, negative numbers refer to terminal nodes, and zeros refer to depths not applicable for this observation.
-----------------------	--

When “Predicted Probabilities” is checked

PROB_<N>	Predicted probabilities (based on the learn data) for each target class.
-----------------------	--

- The predicted probabilities will be included only if the number of classes in the target does not exceed the limit set in the corresponding selection box.

- ✎ All target classes other than the original classes used in learning will be assumed to be missing.

Score GUI Output for Classification Trees

After you click on **[OK]** in the **Score Data** dialog, a progress dialog appears and, after all the cases are dropped down the tree, a **Score** dialog opens and a Text Report appears in the **CART Output** window. The content of both the GUI and the text output for a scoring run will vary depending on whether the target variable is continuous or categorical and whether you are using new or training data. The Score results dialog using a categorical target variable and the GYMTUTOR.CSV training dataset is discussed below. See the subsequent section for a discussion of score output for a regression tree and the CART Reference Manual for a description of Score text reports.

The screenshot shows the 'CART Score: 2' dialog box. It has three tabs: 'Response Statistics', 'Gains', and 'Prediction Success'. The 'Response Statistics' tab is active, displaying a table titled 'Results of Applying CART Tree to Data'. Below the table is an 'Overall Results Summary' section with input fields for 'Train Cases', 'Score Cases', 'Predicted response', and 'Observed response'. The 'Data' field shows 'C:\Program Files\Salford\GYMTUTOR.CSV'.

Node	Cases	Percent Score Data	Percent Train Data	Node Class	Percent Correct
1	71	24.23	24.23	3	100.00
2	4	1.37	1.37	1	100.00
3	7	2.39	2.39	1	100.00
4	89	30.38	30.38	2	95.51
5	25	8.53	8.53	3	80.00
6	12	4.10	4.10	2	91.67
7	1	0.34	0.34	1	100.00
8	2	0.68	0.68	3	100.00
9	82	27.99	27.99	1	100.00

Overall Results Summary

Train Cases: 293 Predicted response: Pct. Correct: 96.59

Score Cases: 293 Observed response:

Grove: Navigator 1 Data: C:\Program Files\Salford\GYMTUTOR.CSV

As illustrated above, the score output dialog displays summary **Response Statistics**, **Gains**, and a **Prediction Success** table for the actual and predicted target variable values. Because the target variable from the original tree appears in the training data, we can access the predictive accuracy of this particular tree.

Response Statistics Tab

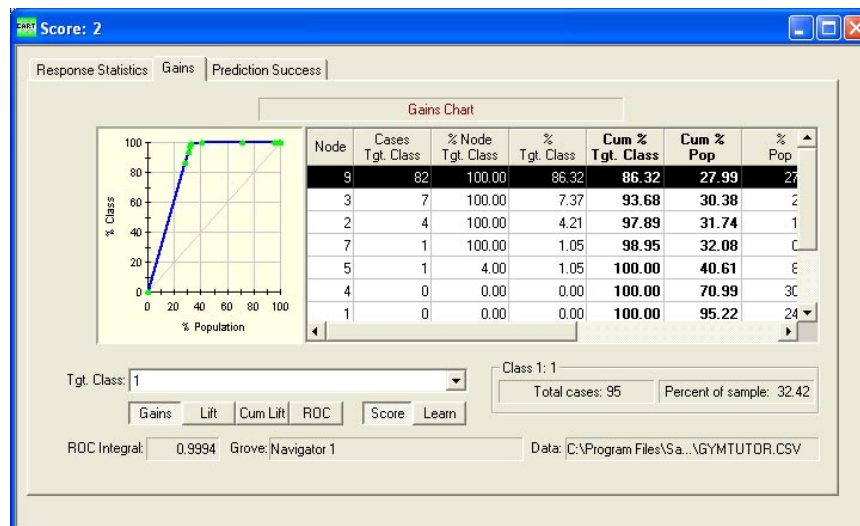
The **Response Statistics** tab provides distributional information by terminal nodes, predicted class, and by the actual target variable (because it is observed). The grid in the top portion displays the following information for each terminal node:

Node	Terminal node number
Cases	Number of cases
Percent Score Data	Percent of scored data in this node
Percent Train Data	Percent of learn data in this node
Node Class	Node class assignment
Percent Correct	Percentage of cases classified correctly in node

The **Results Summary** group box in the lower panel displays the number of predicted cases, the number of observed cases for the target variable, and the percent classified correctly (in this example, 96%). The name of the grove file and the dataset used in the Score run are also noted in the last row of the dialog.

Gains Tab

The **Gains** tab displays gains both in graphical and table forms. Note that you may switch between gains based on the current scored dataset and gains based on the learn data (which define the sort order of the terminal nodes in the gains table).



Prediction Success Tab

The **Prediction Success** tab of the **Score** dialog displays the prediction-success table that cross-classifies the actual by the predicted class (see also the text output for the actual by the predicted node). To view row or column percentages instead of counts, click on **[Row %]** or **[Column %]**.

Score: 2

Response Statistics | Gains | Prediction Success

Results of Applying CART Tree to Data

Actual Class	Total Cases	Percent Correct	Predicted Class		
			1 N=94	2 N=101	3 N=98
1	95	98.95	94	0	1
2	100	96.00	0	96	4
3	98	94.90	0	5	93
Total:		293.00			
Average:		96.62			
Overall % Correct:		96.59			

Count Row % Column %

Grove: Navigator 1 Data: C:\Program Files\...\GYMTUTOR.CSV

Case Output for Regression Trees

To illustrate how to use Score to predict continuous target variables, we will work with the BOSTON.CSV dataset.

First, build the default model using MV as the target and save the resulting navigator as boston.nv3.

Next, press the **[Score...]** button and save the results to a file called "boston_scored.csv." Note that we score the same dataset as was used for learning. If we needed to score another dataset, we would use the **[Select...]** button in the **Data** field to pick the new data file.

Click **[OK]** to start the scoring process.

After all 506 cases are dropped down the tree, a **Score** dialog opens, as shown below, and a Score Text Report appears in the CART Output window.

Score: 2

Response Statistics

Results of Applying CART Tree to Data

Node	Cases	Percent Score Data	Percent Train Data	Predicted Mean	Actual Mean	Train RMS Error	Score RMS Error
1	5	0.99	0.99	45.580	45.580	8.840	8.840
2	43	8.50	8.50	23.970	23.970	1.736	1.736
3	5	0.99	0.99	26.900	26.900	5.685	5.685
4	79	15.61	15.61	19.997	19.997	2.353	2.353
5	68	13.44	13.44	21.659	21.659	2.178	2.178
6	17	3.36	3.36	30.241	30.241	2.319	2.319
7	11	2.17	2.17	24.564	24.564	1.762	1.762
8	18	3.56	3.56	28.500	28.500	2.270	2.270
9	9	1.78	1.78	23.467	23.467	2.158	2.158

Overall Results Summary

Train Cases:	506	Mean Pred. Response:	22.533	RMS Error:	2.534
Score Cases:	506	Mean Obs. Response:	22.533		

Grove: Navigator 1 Data: C:\Program Files\Salford Da...\Boston.csv

Response Statistics Tab

The **Response Statistics** tab provides distributional information by terminal node, predicted response, and observed response. The grid in the top panel displays the following information for each terminal node:

Node	Terminal node number
Cases	Number of cases
Percent Score Data	Percent of scored data in the node
Percent Train Data	Percent of learn data in the node
Predicted Mean	Average of learn cases in the node
Actual Mean	Actual target average in the node
Train RMS Error	RMS error on train data
Score RMS Error	RMS error on scored data

The **Results Summary** group box in the lower panel displays the number of predicted cases, the number of observed cases for the target variable, the predicted response (overall mean for predicted target), the observed response (overall mean for observed target variable), and the total mean squared error for the tree. The name of the grove file and the dataset used are also noted in the last row.

Scoring in Command Line

For command-line scoring the grove file must be saved separately. To score:

- ♦ Issue the GROVE “file_name.grv” command to specify the grove file.
`GROVE <file_name.grv>`
- ♦ Issue one of the following commands to specify other than the optimal tree. If this command is not issued, the optimal tree will be used by default.
`HARVEST PRUNE TREENUMBER=<N>`
`HARVEST PRUNE NODES=<N>`
- ♦ Issue either of the following commands depending on whether or not you want model information added.
`SAVE "<filename>" /MODEL`
`SAVE "<filename>"`
- ♦ Start scoring by issuing
`SCORE [PATH=YES] [PROBS=<N>]`

Translating CART models

Any CART model can be translated into one of the supported languages (SAS[®]-compatible, C, Java, and PMML), or into the classic text output.

The translation operation is very similar to scoring and requires a grove file. As with scoring, you may either use a separate grove file or the one embedded into a navigator.

Translating Using Navigator with Embedded Grove File

The Navigator window must be open and active.

1. Press the **[Translate...]** button.
2. Enter the relevant information into the Model Translation window (described below).
3. Press **[OK]** to activate the translation process.

✂ The **Grove File** portion of the **Model Translation** window will contain your navigator file name—this means that the embedded grove file will be used for translation. You do not have to change this unless you want an external grove file to be used instead.


- This mode is not available with older navigators or navigators that were saved without model information.

Translating Using Grove File Only

If you have a grove file you would like to use for translation, do the following steps:

1. Make sure the CART Output window is active.

Choose **T**ranslate **M**odel... from the **M**odel menu.

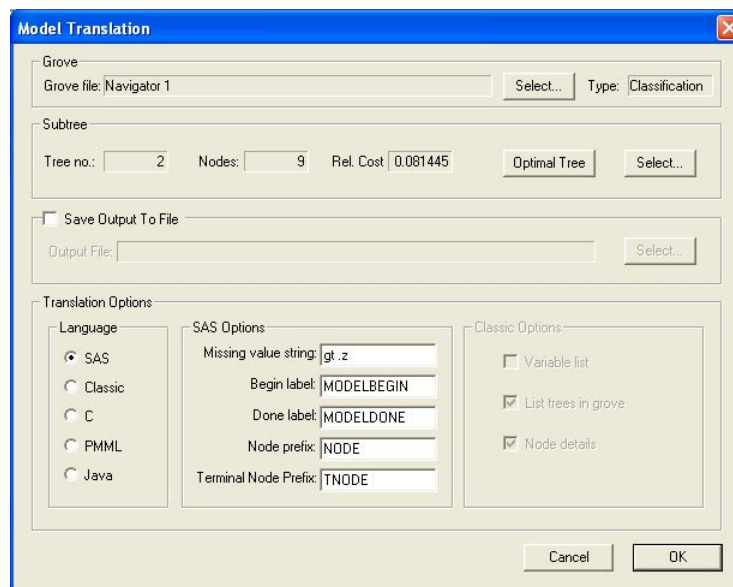
Both these steps can be replaced by simply clicking the  toolbar icon.

Enter relevant information into the **Model Translation** dialog, including the name of the grove file in the Grove File: section.

Press **[OK]** to activate the scoring process.

Model Translation Dialog

The **Model Translation** dialog is shown below:



Grove File

Hit the **[Select...]** button to pick an external grove file for translation or leave this field unchanged if you have a navigator file with an embedded grove file (in which case the navigator file name will appear in the field).

Subtree

Press **[Select...]** to translate other than the optimal tree, then choose the tree in the **Tree Sequence** dialog. By default CART will translate the optimal tree, to which you may always return by pressing the **[Optimal Tree]** button.

Save Output to File

Put a checkmark if you want the results of scoring saved into an external output file. Press the **[Select...]** button to specify the output file's name, location, and extension.

Language

Choose the language; SAS[®]-compatible, Classic, C, PMML, and Java are currently available.

SAS[®]-compatible Options

When translating into SAS, you may also specify additional SAS-related preferences. The definitions should become clear once you look at a sample SAS output.

Classic Options

When translating into Classic, you may further define which pieces of information should be included.

Translating in Command Line

For command-line translating you must have a grove file saved separately. To translate:

1. Issue a GROVE "file_name.grv" command to specify the grove file.
2. Issue one of the following commands to specify other than the optimal tree. If this command is not issued, the optimal tree will be used by default.
 HARVEST PRUNE TREENUMBER=<N>
 HARVEST PRUNE NODES=<N>
3. Depending on the language, issue one of the following commands.
 TRANSLATE LANGUAGE=SAS SMI="gt .z", SBE="MODELBEGIN",
 SDO = "MODELDONE", SNO="NODE", STN="TNODE",
 OUTPUT = "file_name.sas"

 TRANSLATE LANGUAGE=CLASSIC, VLIST=YES, TLIST=YES,
 DETAILS=YES, SURROGATES=YES,
 OUTPUT = "<file_name>"

 TRANSLATE LANGUAGE=C OUTPUT="<file_name.c>"

 TRANSLATE LANGUAGE=PMML OUTPUT="<file_name.xml>"

Exporting and Printing Tree Rules

The scoring procedure described above generates detailed output. The resulting code includes not only main splitters, but also all surrogate splits as alternative conditions. While having this code is invaluable for external scoring, it might be overkill if all you need is a set of simple rules based on main splitters only.

CART 6 has inherited an older simplified version of model translation that is accessible by selecting **Rules...** from the **View** menu when the current navigator is active.

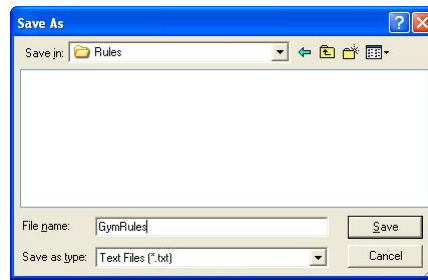
- ✎ Alternatively, you may right-click on the root node and select **Rules** from the local menu.
- ✎ You may also generate rules for only a branch of a tree by right-clicking on the internal node that originates the branch and selecting **Rules** from the local menu.

To add the within-node probabilities for the learn or test samples, click **[Learn]** or **[Test]**. Combined learn and test probabilities can be added by clicking **[Pooled]**. For example, the main tree rule dialog for the GOODBAD.CSV dataset with learn sample probabilities activated is displayed below.



You can also view the rules for a particular tree node. In the Navigator, click on the node of interest and select the **Rules** tab from the terminal node results dialog.

To export rules as a text file, select **Export...** from the **File** menu. In the **Save As** dialog, specify a directory and file name; the file extension is by default *.txt*.



To send the rules to the printer, select **Print** from the **File** menu when the Main Tree dialog is active. You can also copy and paste the rules onto the clipboard or directly into another application.

Train-Test Consistency (TTC)

*A new powerful feature designed
to identify stable robust trees*



Optimal Models and Tree Stability

CART relies on the concept of pruning to create a sequence of nested models as final model candidates. Independent testing or cross validation is subsequently used to identify the optimal tree with respect to overall model performance (based on the expected cost criterion). This, however, does not guarantee that the resulting tree will show stable performance at the node level. It is quite possible to have a node created earlier in the tree exhibiting unstable behavior across different partitions of the data. Often such nodes cannot be easily eliminated without picking a much smaller tree in the pruning sequence, thus picking an inferior (in terms of accuracy) model. Nonetheless, some analysts might be more interested in finding robust trees with all nodes exhibiting stable behavior and be less concerned with the actual accuracy measures (for example, marketing segmentation problems). The TTC feature of CART was designed to facilitate such an analysis of the tree sequence.

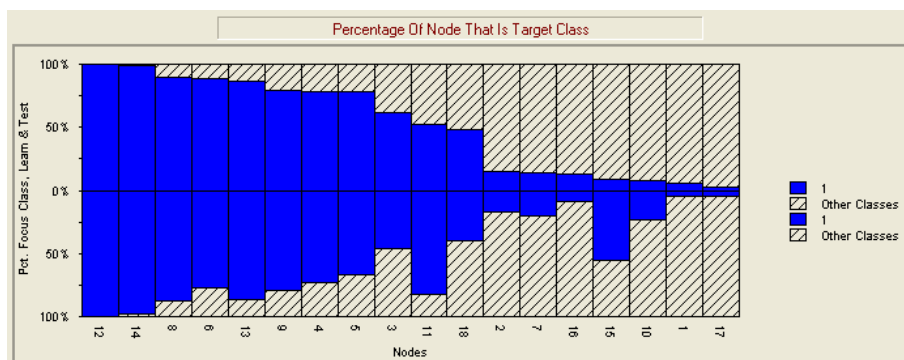
Spam Data Example

We illustrate the specifics of the TTC feature using the SPAMBASE.CSV dataset.

First, use the **Open->Command File...** option from the **File** menu to open the TTC.CMD command file.

Second, use the **File->Submit Window** menu to build a new model. The resulting Navigator suggests an 18-node tree as the optimal in terms of expected cost.

Now press the **[Summary Reports...]** button and go to the **Terminal Nodes** tab.



Note two types of instability of the optimal tree with respect to the Learn and Test results:

Directional Instability – Node 15 has 9% of Class=1 on the learn data and 56% of Class=1 on the test data. Assuming the node majority rule for class assignment, this effectively constitutes instability with respect to the class assignment that depends on the data partition. Another way to look at this is that the node lift is less than 1 on the learn data and greater than 1 on the test data.

Rank Instability – The nodes on the graph are sorted according to node richness using the learn data. However, the sorted order is no longer maintained when looking at the test data; hence, we have another type of instability. Many deployment strategies (for example, model-guided sampling of subjects in a direct marketing campaign) rely only on the sorted list of segments and therefore eliminating this kind of instability is highly desirable.

Note that the Rank Stability requirement is generally stricter than the Directional Stability requirement. In other words, one may have all nodes directionally stable (agree on the class assignment) and yet have non-conforming sort orders.

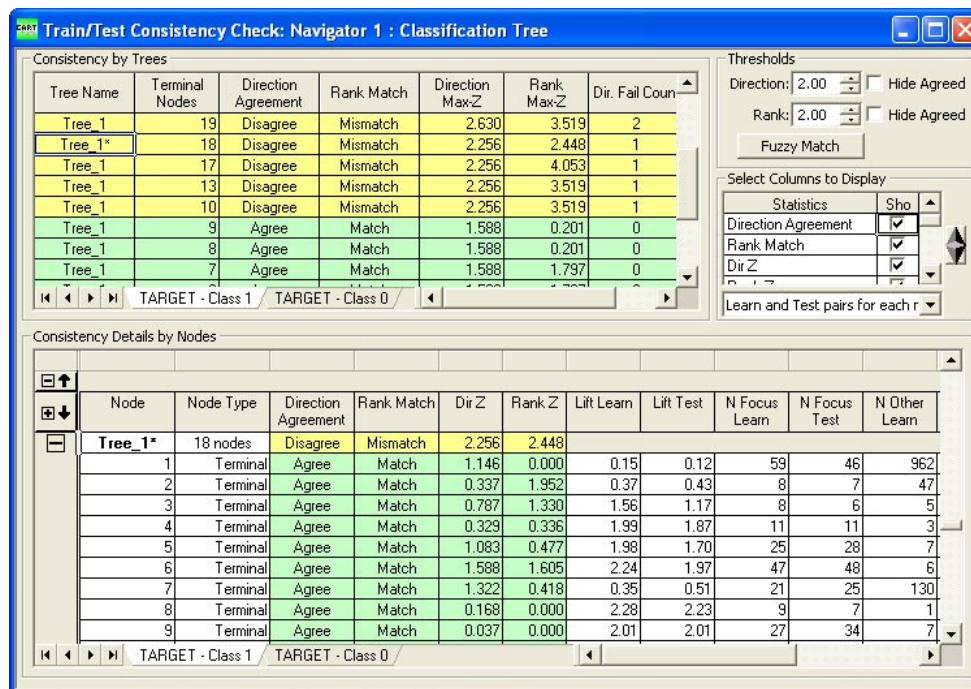
Also note that it is useful to introduce some “slack” in the above comparisons due to limited node sizes. For example, one might argue that the discrepancies in the sort sequences must be significant enough to declare the whole model as rank unstable. Similarly, a directional disagreement node must show a significant difference between the learn and test sample estimates. We employ a simple statistical test on a difference in two population proportions to accomplish this. The z-threshold of this test is controlled by the user, thus giving varying degrees of slack.

In addition, special care must be taken in handling nodes where the test data is missing entirely (empty test counts). The user has the option to either declare such trees unstable or to ignore any such node (Fuzzy Match) .

Running TTC

To run a TTC analysis just press on the **[T/T Consist...]** button in the Navigator Window.

The resulting display shows the results:



The upper half reports stability by trees, one line per tree. You can choose the class of interest by clicking on the corresponding tab. Green marks stable trees while yellow marks unstable trees. Note that because there are two different approaches to tree stability (rank or directional), it is possible to have a tree agree on one criterion and disagree on the other.

The columns in the **Consistency by Trees** section are:

Tree Name – name of the tree. It is a constant for single trees but will have varying values for batteries of CART runs (when applicable).

Terminal Nodes – number of terminal nodes.

Direction Agreement – contains “Agree” if all terminal nodes agree on the direction of classification (within the supplied degree of confidence).

Rank Match – contains “Agree” if all terminal nodes agree on the sorted sequence as described above.

Direction Max-Z – reports the z-value of the standard statistical test on the difference in two population proportions – learn node content versus test node content. Note that a node may agree on the direction (class assignment) but still

have a significant difference between the learn and test proportions as reflected by the z-value.

Rank Max-Z – reports the z-value of the standard statistical test on the difference in two population proportions as follows. We first sort nodes by the learn-based responses, then we sort nodes by the test-based responses, and finally we look at the nodes side by side and check the difference in test-based proportions for each pair.

Dir. Fail Count – reports the total number of terminal nodes in the tree that failed directional agreement.

Rank Fail Count – reports the total number of terminal node pairs in the tree that failed the rank agreement.

The **Consistency Details by Nodes** (lower half) provides a detailed node-by-node stability report for the tree selected in the Consistency by Trees part (upper half). For example, the optimal tree with 18 terminal nodes has one directional instability in node 15 (as seen by scrolling the list in the lower half) at the given significance level.

In addition to the columns already present in the Consistency by Trees report, the following ones are added:

Lift Learn – node lift on the train data

Lift Test – node lift on the test data

N Focus Learn – number of train records that belong to the focus class in the node

N Focus Test – number of test records that belong to the focus class in the node

N Other Learn – number of train records that do not belong to the focus class in the node

N Other Test – number of test records that do not belong to the focus class in the node

N Node Learn – number of train records in the node

N Node Test – number of test records in the node

You can control which columns are shown and in what order in the **Select Columns to Display** section.

The following group of controls allows fine user input:

The image shows a 'Thresholds' dialog box with the following controls:

- Direction:** A numeric input field set to 2.00, with a 'Hide Agreed' checkbox to its right.
- Rank:** A numeric input field set to 0.50, with a 'Hide Agreed' checkbox to its right.
- Fuzzy Match:** A button located below the input fields.

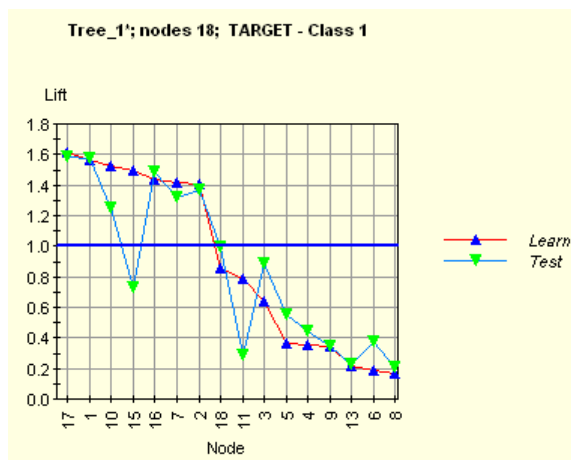
Direction – sets the z-value threshold on the directional stability. A node is declared directionally unstable only if it has contradicting class assignments on learn and test samples and furthermore has the z-value of the corresponding test greater than the threshold. Otherwise, the node is directionally stable (has identical class assignments or z-value is below the threshold).

Rank – sets the z-value threshold on the rank stability. A pair of nodes (taken from learn- and test-based sorted sequences) is declared rank stable if the z-value of the corresponding test is below the threshold.

Fuzzy Match – determines whether empty nodes (on test data) are ignored (Fuzzy Match is pressed) or treated as unstable ([**Fuzzy Match**] is not pressed).

Hide Agreed – hides all agreed terminal nodes from the Consistency Details by Nodes report.

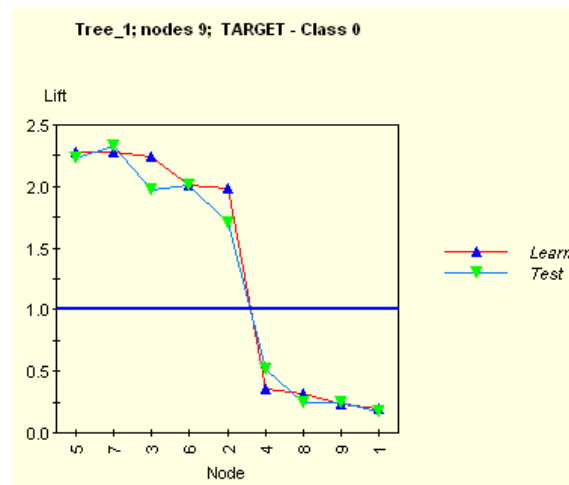
Double-clicking on any tree in the Consistency by Trees section (upper half) will result in a graph of train and test focus class lift by node.



Note the apparent directional instability of Node 15 (Learn and Test values are on the opposite sides of the 1.0 lift curve) as well as the rank instability of the Test curve (severe deviation from monotonicity).

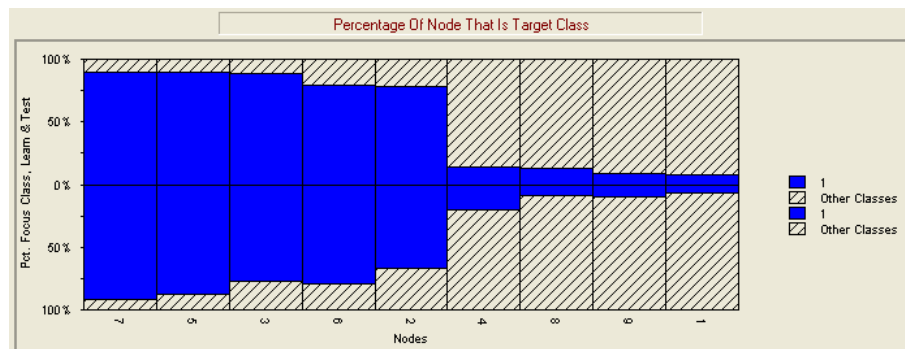
Identifying Stable Tree

Now let us use the TTC results to identify a consistent tree. As can be seen in the Consistency by Trees table, the 9-node tree is stable both in direction and rank.



Note that even though the rank stability is approximate (slight departures from monotonicity in the Test curve), it is well within the significance level controller by the Rank z-threshold.

Summary Reports – Terminal Nodes further illustrates the tree stability we were initially looking for.



Hot Spot Detection

*A new powerful feature designed to identify
hot spots in the class of interest*



Searching for Hot Spots

In many modeling situations an analyst is looking for regions of modeling space richest in the event of interest. These regions are usually called Hot Spots. For example, in fraud detection problems, we could be interested in identifying a set of rules that lead to a high ratio of fraud so as to flag records that are almost guaranteed to be fraudulent. Because target classes usually overlap (making it impossible to have a clear separation of one target group from the other), a search for hot spots usually results in a reduced overall accuracy in the class of interest. In other words, while it might be possible to identify areas of data rich in the event of interest, chances are that a substantial amount of data will be left outside the covered areas that cannot be easily separated from the remaining class.

One of the advantages of CART is that it gives clear sets of rules describing each terminal node. Therefore, searching for hot spots usually boils down to searching for nodes richest in the given class across multiple trees. The hot spot machinery described below can be applied to a single tree, but it is most beneficial in processing CART battery models (collections of trees obtained by a systematic change in model settings). While any CART battery can be used, the most suitable for the task is battery prior. This battery varies the prior probabilities used in tree construction, thus directly enforcing different requirements in the tradeoff between node richness and class accuracy.

Spam Data Example

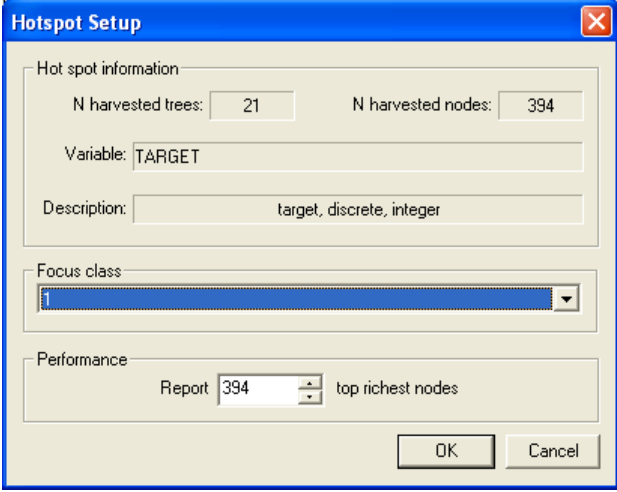
We illustrate searching for hot spots using the SPAMBASE.CSV dataset as an example.

First, use the **Open->Command File...** option from the **File** menu to open the HOTSPOT.CMD command file. Note at the bottom of the command file that we will be running battery prior with priors on class 1 (spam group) varying between 0.5 and 0.9 in increments of 0.02, thus producing 21 models.

Second, use the **File->Submit Window** menu to build the battery. The resulting Battery Summary contains information on all 21 models requested. Our goal is to scan all terminal nodes across all models and identify the nodes richest in spam.

✂ In CART 6.0 we introduced the modeling automation technique known as "batteries." This feature, discussed in the following chapter, makes the process of modeling in batches as easy as a mouse click.

Form the **Report** menu, select **Gather Hotspot...** which gives us the following **HotSpot Setup** dialog.



Hotspot Setup

Hot spot information

N harvested trees: 21 N harvested nodes: 394

Variable: TARGET

Description: target, discrete, integer

Focus class:

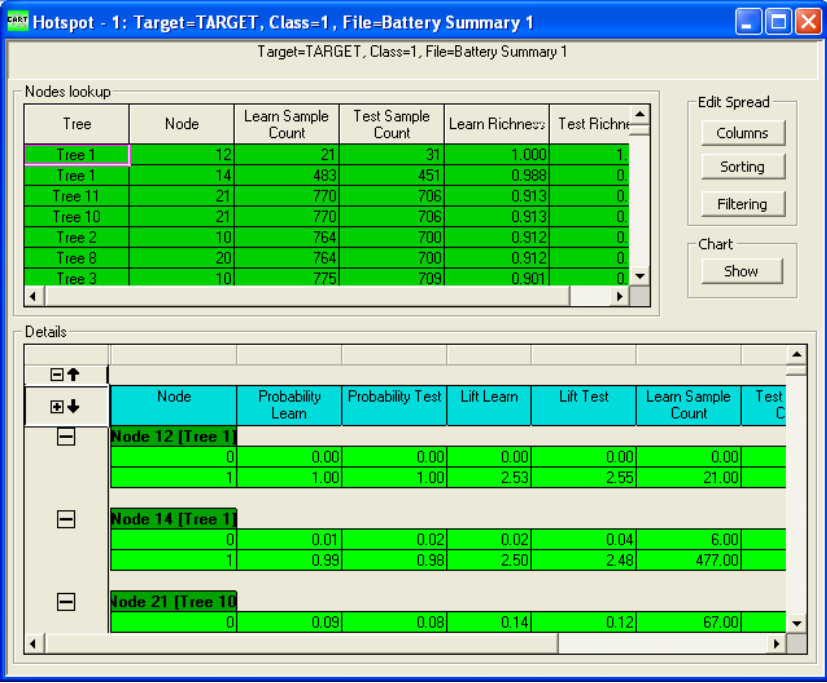
Performance

Report 394 top richest nodes

OK Cancel

Note that there are 394 terminal nodes across 21 trees in the battery. We also set Focus class to 1 (spam group) and request actual processing of the entire pool of terminal nodes. Pressing **[OK]** will produce two windows: **Hotspot Table** and **Hotspot Chart**.

The Hotspot Table window contains the results of hotspot analysis in tabular form.



Hotspot - 1: Target=TARGET, Class=1, File=Battery Summary 1

Target=TARGET, Class=1, File=Battery Summary 1

Nodes lookup

Tree	Node	Learn Sample Count	Test Sample Count	Learn Richness	Test Richness
Tree 1	12	21	31	1.000	1
Tree 1	14	483	451	0.988	0
Tree 11	21	770	706	0.913	0
Tree 10	21	770	706	0.913	0
Tree 2	10	764	700	0.912	0
Tree 8	20	764	700	0.912	0
Tree 3	10	775	709	0.901	0

Details

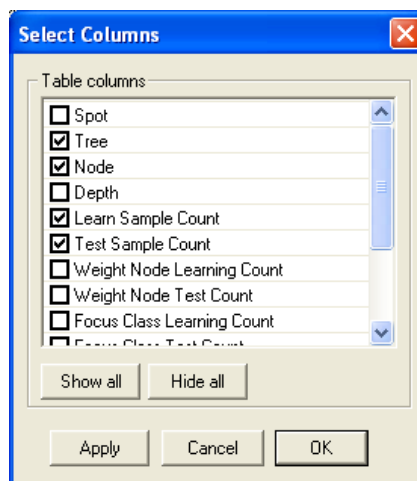
Node	Probability Learn	Probability Test	Lift Learn	Lift Test	Learn Sample Count	Test Sample Count
Node 12 [Tree 1]						
0	0.00	0.00	0.00	0.00	0.00	0.00
1	1.00	1.00	2.53	2.55	21.00	31.00
Node 14 [Tree 1]						
0	0.01	0.02	0.02	0.04	6.00	4.00
1	0.99	0.98	2.50	2.48	477.00	447.00
Node 21 [Tree 10]						
0	0.09	0.08	0.14	0.12	67.00	60.00

The upper **Nodes lookup** table contains all requested terminal nodes (one line per node) sorted according to learn node richness.

The default columns are:

- ◆ **Tree** – unique tree identifier in the current battery
- ◆ **Node** – unique terminal node identifier in the current tree
- ◆ **Learn Sample Count** – node size on the train data
- ◆ **Test Sample Count** – node size on the test data
- ◆ **Learn Richness** – node richness in the focus class on the train data (using this column, table rows are sorted descending)
- ◆ **Test Richness** – node richness in the focus class on the test data

In addition, the **[Columns]** button in the **Edit Spread** group of controls allows the selective addition of more columns to the table:



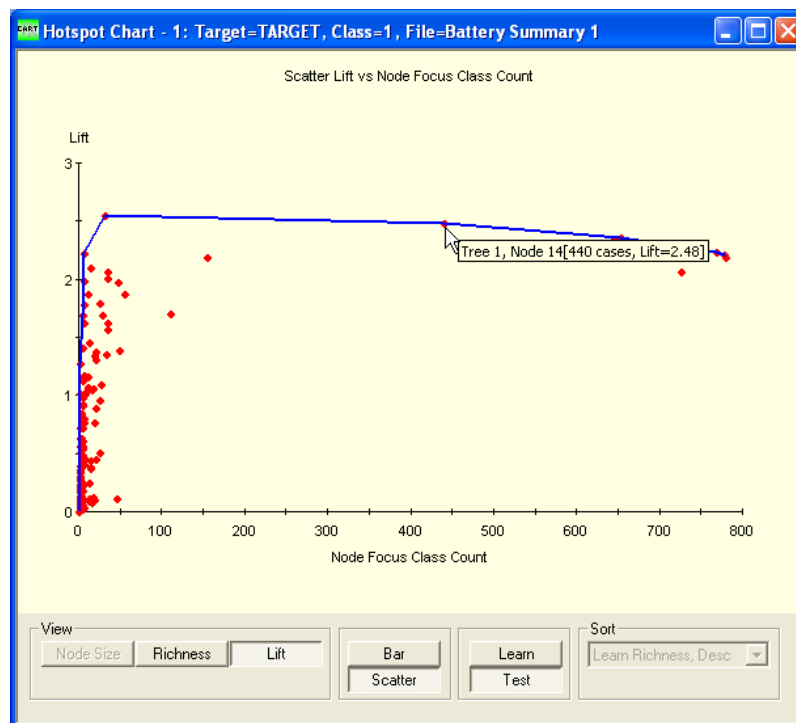
- ◆ **Spot** – sequential hotspot identifier
- ◆ **Depth** – depth level of the node in the tree
- ◆ **Weight Node Learning Count** – weighted node size on the train data
- ◆ **Weight Node Test Count** – weighted node size on the test data
- ◆ **Focus Class Learning Count** – number of focus class records in the node on the train data
- ◆ **Weight Focus Class Learning Count** – same as above but weighted
- ◆ **Focus Class Test Count** – number of focus class records in the node on the test data
- ◆ **Weight Focus Class Test Count** – same as above but weighted

You can change the default sorting method of the nodes using the **[Sorting]** button in the **Edit Spread** group or introduce your own filtering conditions using the **[Filtering]** button in the same group.

The lower **Details** part of the table contains additional information on each terminal node, including not only the focus class but also all the remaining classes.

According to the table, Node 12 of Tree 1 has 100% test richness but only 31 cases. Node 14 of the same tree is 97.6% rich on a much larger set of 451 test cases. An even larger node (706 test cases) is found in Tree 11, which has a reduced richness of 92.5%. You can double click on any of the nodes to request the corresponding navigator window to show up.

Pressing on the **[Show]** button brings up the **Hotspot Chart** window:



The graph shows a scatter plot of node richness (or node lift when the corresponding button is pressed) versus node focus class count. You can switch between the **[Bar]** and **[Scatter]** views of the plot. You can also switch between the **[Learn]** and **[Test]** results.

Hovering the mouse pointer over a dot produces extra information that contains tree and node number as well as the actual coordinate values as shown above.

Finally, the blue line marks the “Effective Frontier” —the nodes most interesting in terms of balancing node richness versus node size.

Chapter

10

CART Batteries

*A new and powerful feature designed
to build multiple models automatically*

Batteries of Runs

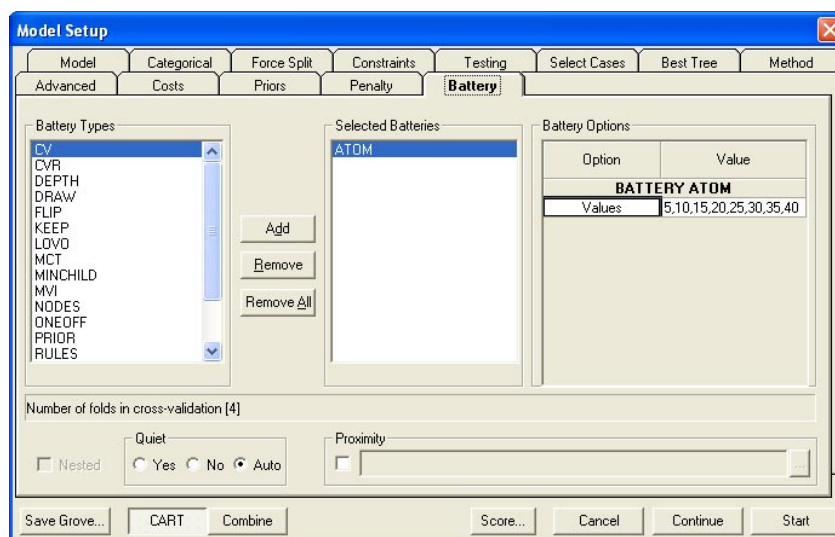
The CART algorithm is characterized by a substantial number of control settings. Often the optimal values for many parameters cannot be determined beforehand and require a trial and error experimental approach. In other cases, it is desirable to try various settings to study their impact on the resulting models. CART batteries were designed to automate the most frequently-occurring modeling situations that require multiple collections of CART runs.

We start our discussion with a description of common battery controls and output reports. Then we move on to a detailed description of each of the available batteries and highlight the specifics of their use.

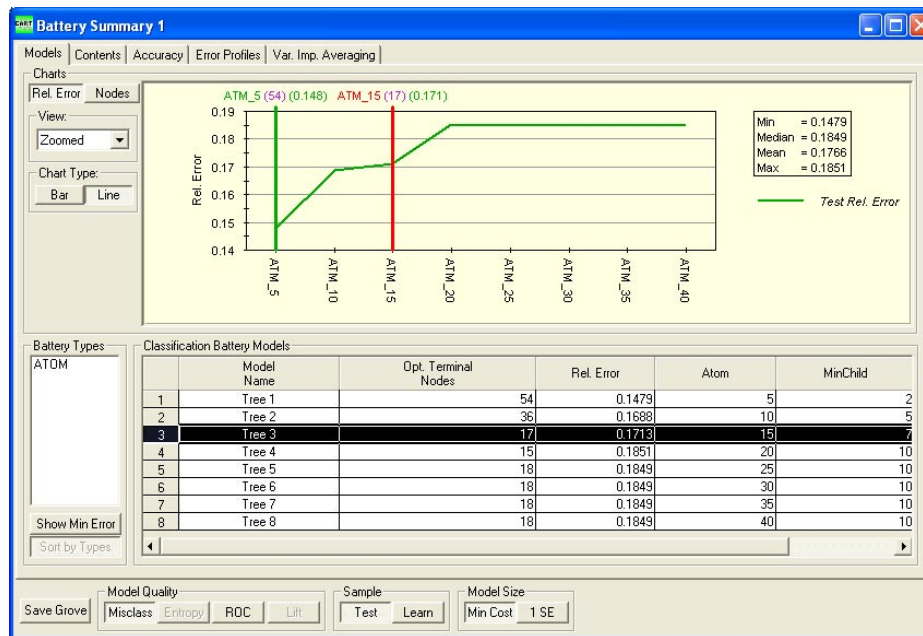
Common Battery Controls

A large number of batteries are simply collections of runs where one specific control parameter is set to different values. You access battery setup from the **Battery** tab in the **Model Setup** menu.

Consider, for example, battery ATOM that varies the atom size (the minimum required parent node size; see the ATOM.CMD command file).



First, highlight ATOM in the **Battery Types** selection panel and press the **[Add]** button. Then type in a list of possible atom values in the **Values** entry box found in the panel titled "Battery Options." Pressing the **[Start]** button produces a summary report window for the resulting eight models with different atom settings:



The relative error of the optimal model obtained in each run is shown on the upper graph within the Battery Summary—Models tab. You can view the graph in the **[Line]** or **[Bar]** styles, **Zoomed** or **All Models**, as well as **[Rel. Error]** or **[Nodes]** for the Y-axis. When the **[Show Min Error]** button is pressed, the model having the smallest relative error is highlighted in green.

Model performance can be viewed in terms of relative error (**[Misclass]** button) or the average area under the ROC curve (**[ROC]** button). Furthermore, it can be presented on the test data (default, **[Test]** button) or on the train data (**[Learn]** button). It is also possible to switch between optimal minimum cost trees (**[Min Cost]** button) and 1-SE trees (**[1 SE]** button). A 1-SE tree is defined as the smallest tree with the relative cost within one standard deviation of the optimal smallest relative cost tree).

The **Classification Battery Models** section in the lower half contains a tabular depiction of the results with the following columns:

Model Name – unique model identifier

Opt. Terminal Nodes – number of terminal nodes in the optimal (minimum relative error) tree (when **[Min Cost]** is pressed)

1 SE Terminal Nodes – number of terminal nodes in the 1 SE tree (when **[1 SE]** is pressed)

Rel. Error – relative error of the model (when **[Misclass]** is pressed)

Avg. ROC – average area under the ROC curve (when **[ROC]** is pressed)

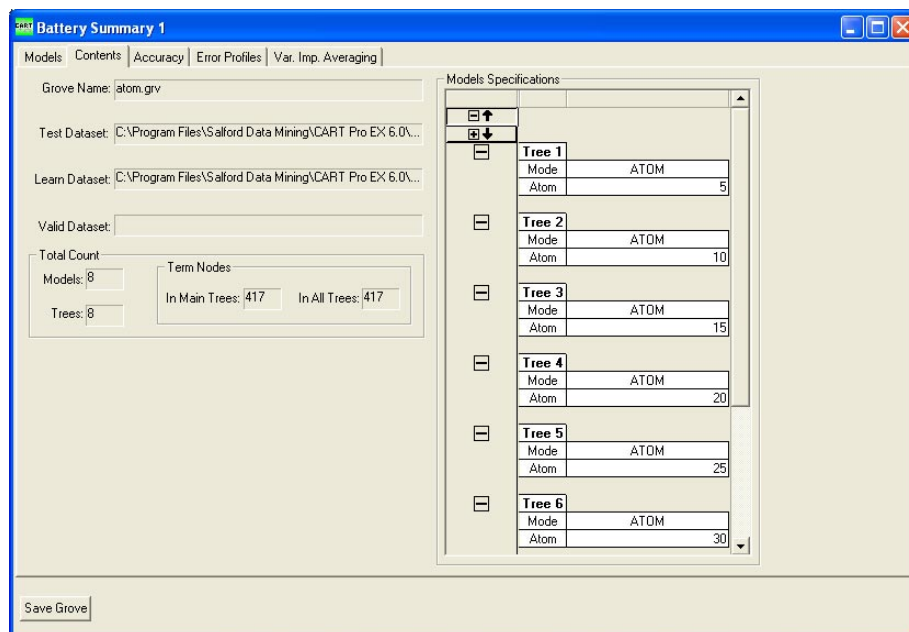
Atom – minimum required parent node size

MinChild – minimum required terminal node size

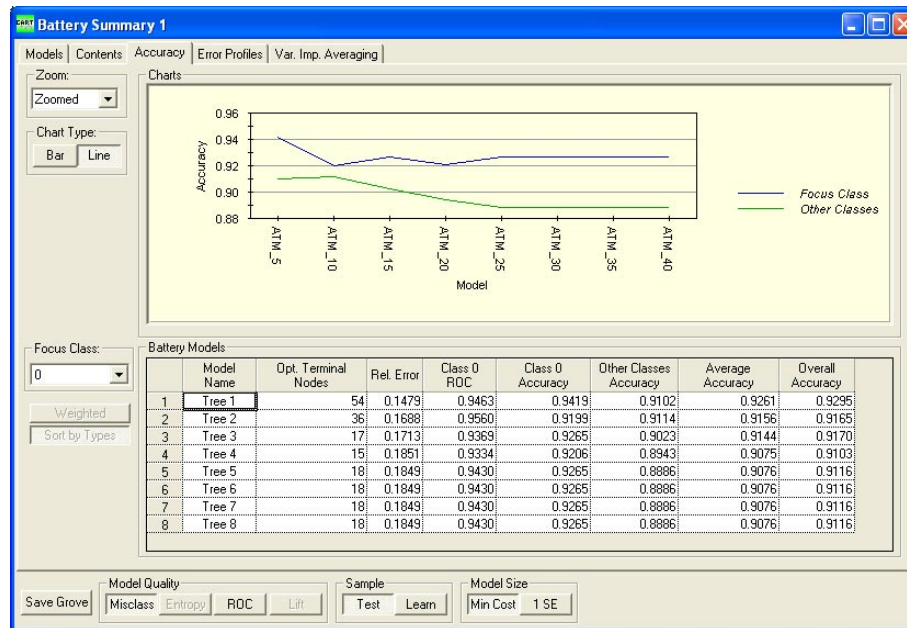
Double-click on any line in the Classification Battery Models section to open the corresponding Navigator window.

The entire battery can be saved using the **[Save Grove]** button.

The **Contents** tab includes summary information about the battery as well as a battery-specific description of each individual model in the **Models Specifications** section:



The **Battery Summary—Accuracy** tab further extends information previously presented on the **Models** tab:



The upper graph shows the accuracy of the focus class (blue curve) and accuracy in the remaining classes (green curve) by models. The table below contains the actual values in the following columns:

Model Name – unique model identifier

Opt. Terminal Nodes – number of terminal nodes in the minimum cost tree ([**Min Cost**] is pressed)

1 SE Terminal Nodes – number of terminal nodes in the 1SE tree ([**1 SE**] is pressed)

Rel. Error – relative error

Avg. ROC – average area under the ROC curve

Class 0 ROC – the ROC for the class in focus (the **Focus Class** selection box controls which class is put in focus, class 0 in this example)

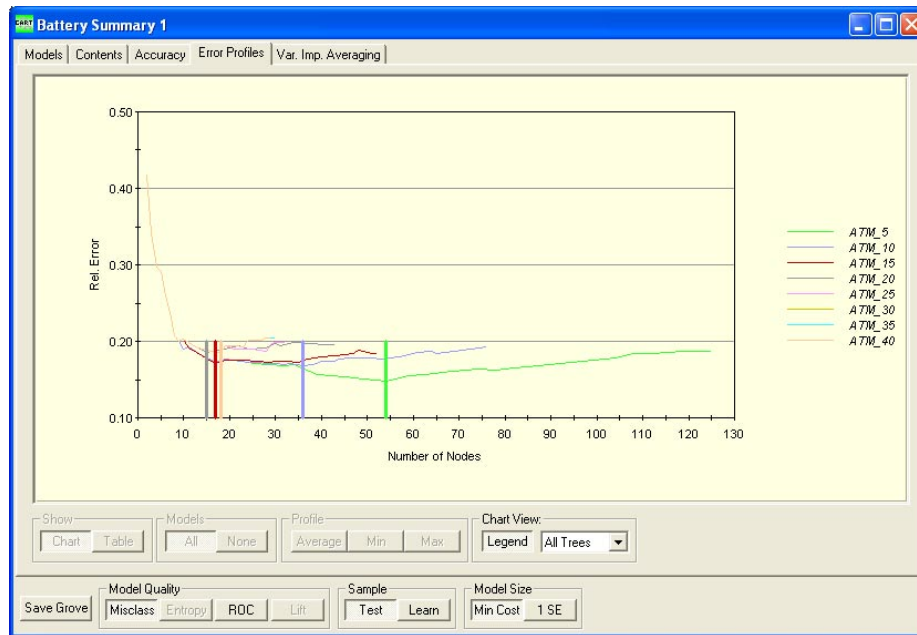
Class 0 Accuracy – accuracy in the focus class

Other Classes Accuracy – accuracy in the remaining classes

Average Accuracy – average accuracy over all available classes

Overall Accuracy – overall accuracy of the model

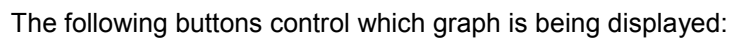
The **Battery Summary—Error Profiles** tab shows the actual model profiles for each run:



Relative error profiles are shown when the **[Misclass]** button is pressed. The areas under the ROC profiles are shown when the **[ROC]** button is pressed. You can also switch between **[Learn]** and **[Test]** profiles. The vertical markers indicate the optimal tree positions (**[Min Cost]** button is pressed) or 1SE tree positions (**[1 SE]** button is pressed).

The legend can be turned on or off using the **[Legend]** button.

Finally, the **Battery Summary—Var. Imp. Averaging** tab shows the results of variable importance averaging across all models in the battery:



- ◆ **[Min]** – smallest importance value for the variable across all models
- ◆ **[Quartile 0.25]** – first quartile importance value across all models
- ◆ **[Median]** – median (second quartile) importance value across all models
- ◆ **[Quartile 0.75]** – third quartile importance value across all models
- ◆ **[Max]** – maximum importance value across all models
- ◆ The sort order of variables can be changed using the **Sort:** selection box.

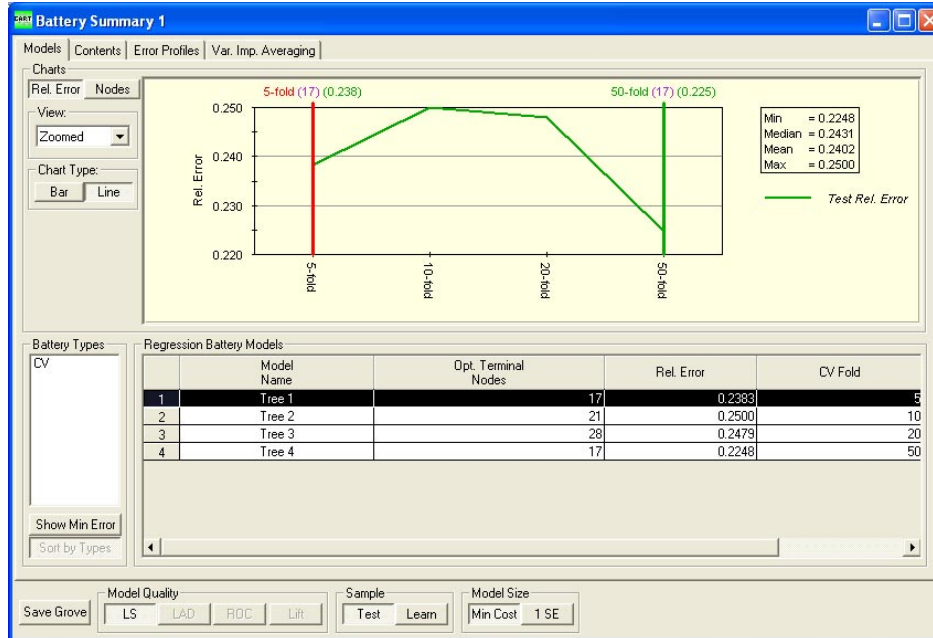
- ◆ **[Box Plot]** – shows a box plot of importance scores for each variable
- ◆ **[Mean]** – shows a mean importance profile
- ◆ **[Grid]** – adds a grid to the display

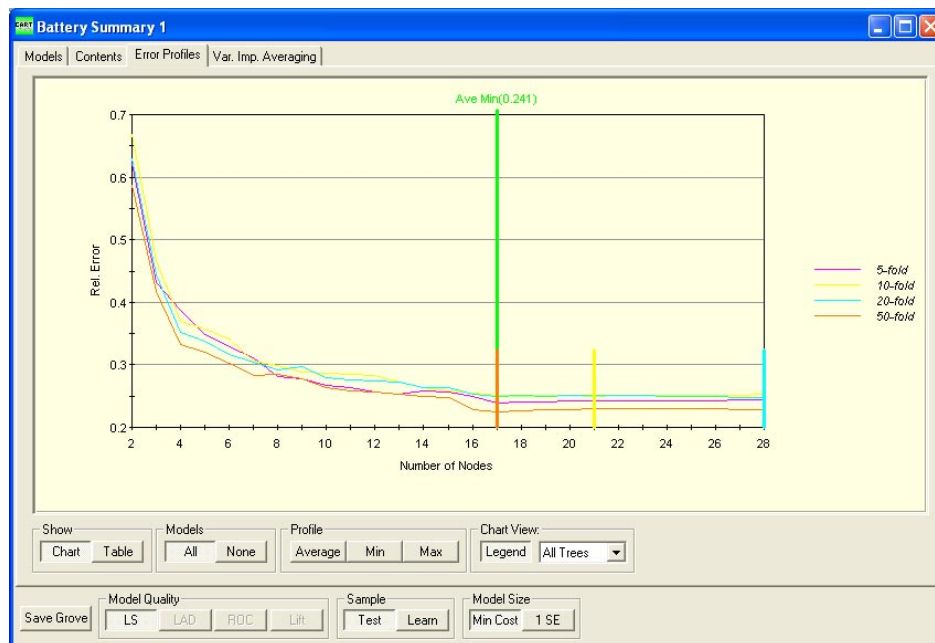
We now proceed with the description of all available batteries.



Battery CV

Battery CV runs cross validation with the number of folds set to 5, 10, 20, and 50 bins. See the CV.CMD command file for run details on the BOSTON.CSV dataset.

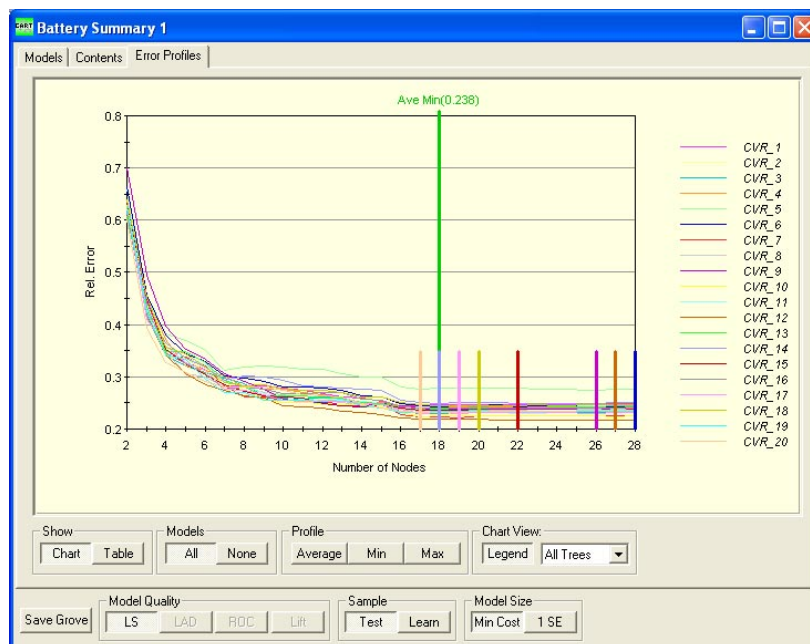






Battery CVR

Battery CVR repeats cross validation many times using different random number seeds. We illustrate it on the BOSTON.CSV dataset by requesting 20 cycles (see CVR.CMD command file for details):



Note that the **Var. Imp. Averaging** tab is no longer available because each individual run has the same master sequence and resulting variable importance.

In addition to the actual run profiles, you may add average (**[Average]** button), minimal (**[Min]** button) and maximal (**[Max]** button) profiles. It is also possible to hide individual profiles using the **[None]** button.

You can switch from the chart view (**[Chart]** button) to the table view (**[Table]** button). In the table view, columns represent relative error sequences for each model. Optimal trees are highlighted in green while 1-SE trees are highlighted in pink.

Battery Summary 1											
Models Contents Error Profiles											
Number of Nodes	CVR_1 Rel. Error	CVR_2 Rel. Error	CVR_3 Rel. Error	CVR_4 Rel. Error	CVR_5 Rel. Error	CVR_6 Rel. Error	CVR_7 Rel. Error	CVR_8 Rel. Error	CVR_9 Rel. Error	CVR_10 Rel. Error	CVR_11 Rel. Error
2	0.6187	0.6331	0.6199	0.6630	0.6603	0.6636	0.6105	0.6375	0.7011		
3	0.4428	0.4501	0.4227	0.4537	0.4511	0.4575	0.4244	0.4202	0.4946		
4	0.3533	0.3681	0.3392	0.3719	0.3825	0.3798	0.3407	0.3476	0.3972		
5	0.3277	0.3388	0.3216	0.3207	0.3687	0.3448	0.3379	0.3067	0.3518		
6	0.3140	0.3153	0.3084	0.3118	0.3499	0.3291	0.3204	0.2998	0.3338		
7	0.2821	0.2837	0.2980	0.3011	0.3126	0.3010	0.2836	0.2812	0.3064		
8	0.2827	0.2741	0.2866	0.2801	0.3195	0.2967	0.2608	0.2801	0.2904		
9	0.2827	0.2649	0.2616	0.2772	0.3201	0.2901	0.2605	0.2703	0.2753		
10	0.2784	0.2604	0.2586	0.2768	0.3194	0.2804	0.2542	0.2652	0.2578		
11	0.2732	0.2596	0.2580	0.2698	0.3155	0.2799	0.2530	0.2578	0.2531		
12	0.2671	0.2596	0.2607	0.2666	0.3145	0.2783	0.2513	0.2547	0.2511		
13	0.2611	0.2526	0.2585	0.2579	0.3054	0.2740	0.2527	0.2515	0.2462		
14	0.2583	0.2528	0.2511	0.2519	0.2990	0.2636	0.2503	0.2454	0.2455		
15	0.2525	0.2497	0.2457	0.2510	0.2991	0.2615	0.2479	0.2397	0.2479		
16	0.2455	0.2429	0.2345	0.2435	0.2810	0.2489	0.2283	0.2378	0.2443		
17	0.2450	0.2384	0.2305	0.2386	0.2773	0.2445	0.2227	0.2345	0.2396		
18	0.2461	0.2365	0.2292	0.2405	0.2777	0.2430	0.2225	0.2369	0.2371		
19	0.2462	0.2378	0.2313	0.2418	0.2782	0.2445	0.2222	0.2380	0.2380		
20	0.2467	0.2400	0.2318	0.2418	0.2786	0.2445	0.2244	0.2370	0.2371		
21	0.2476	0.2409	0.2331	0.2427	0.2781	0.2458	0.2247	0.2369	0.2381		
22	0.2471	0.2413	0.2333	0.2420	0.2777	0.2432	0.2253	0.2369	0.2377		
23	0.2472	0.2411	0.2333	0.2420	0.2768	0.2438	0.2252	0.2375	0.2377		

Show:
Models:
Profile:
Chart View:

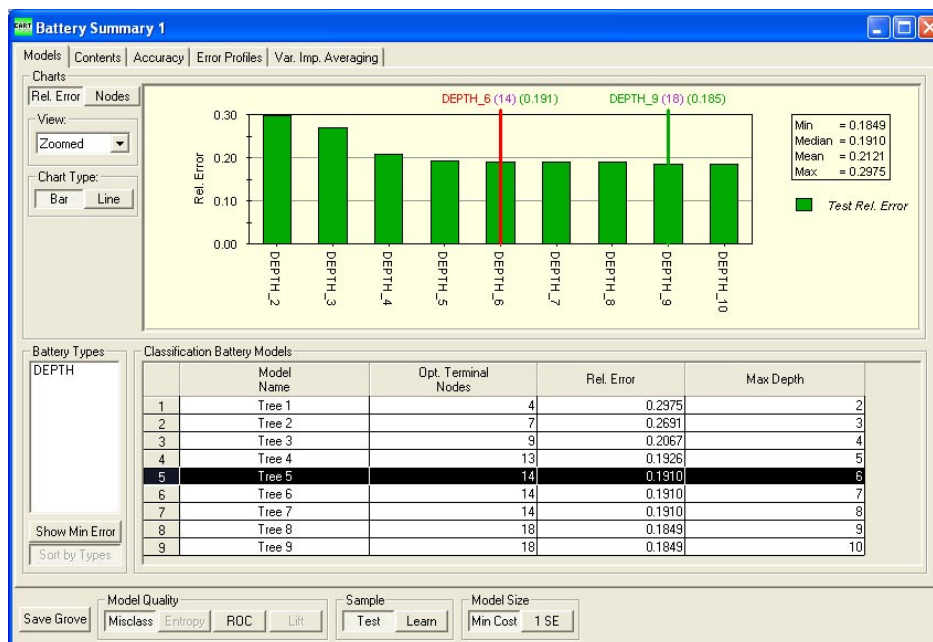
Save Grove:
Sample:
Model Size:

According to our findings, the relative CV-error could be as low as 0.216 or as high as 0.275 with the average at 0.238.



Battery DEPTH

Battery DEPTH specifies the depth limit of the tree. We illustrate it on the SPAMBASE.CSV dataset by trying depths at 2, 3, 4, 5, 6, 7, 8, and 9 (see DEPTH.CMD command file for details):



Clearly, beginning at the depth of 6, the relative error becomes quite flat.

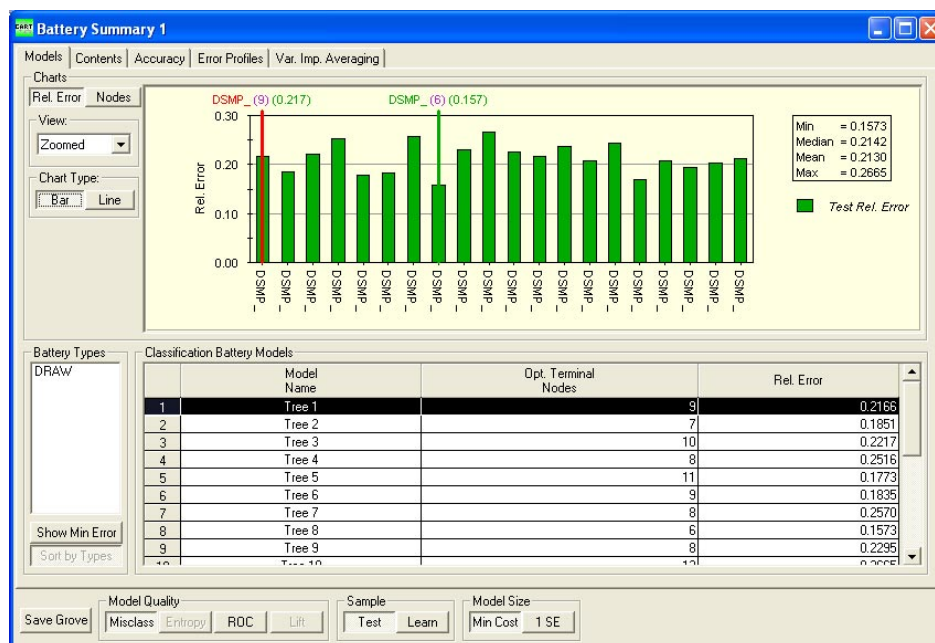


Battery DRAW

Battery DRAW runs a series of models where the learn sample is repeatedly drawn (without replacement) from the “main” learn sample as specified by the Testing tab. The test sample is not altered.

This battery is useful for determining the impact of varying random learn sample selection on ultimate model performance. This is similar in spirit to the battery CVR described earlier.

We illustrate this battery on the SPAMBASE.CSV dataset partitioned into 70% learn and 30% test, with twenty 50% drawings from the learn partition (see DRAW.CMD command file):



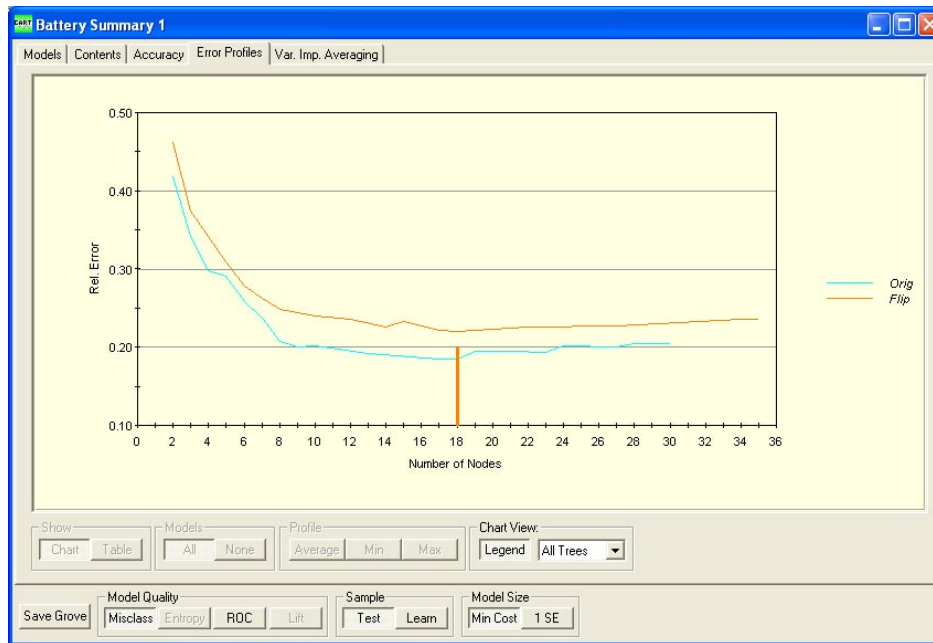
As the results indicate, the effect of sampling the learn data alone produces relative errors between 0.1573 and 0.2665.



Battery FLIP

Battery FLIP generates two runs with the meaning of learn and test samples flipped. The user has to specify the test sample explicitly using the Testing tab in the Model Setup window.

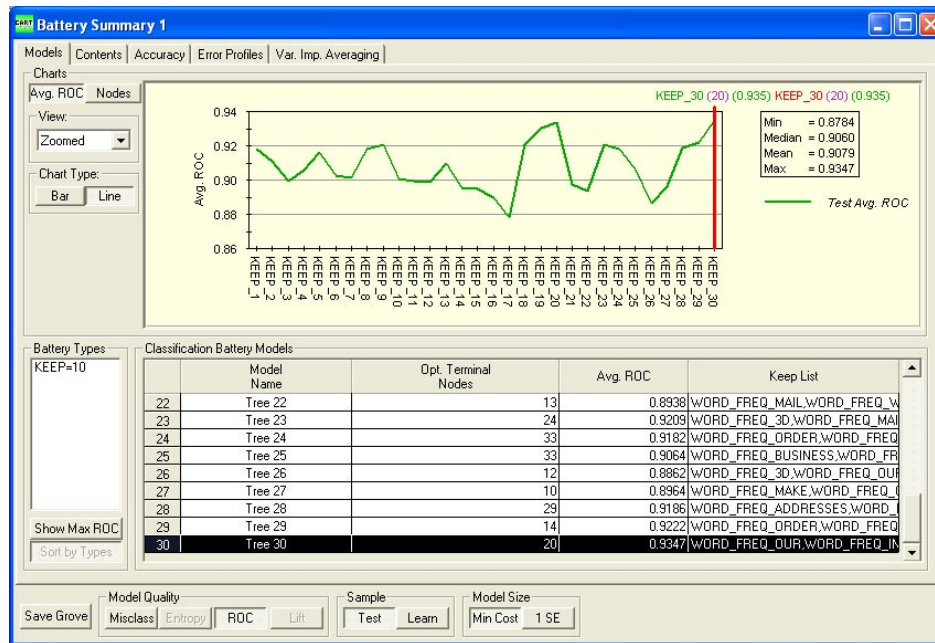
We illustrate the use of this battery on the SPAMBASE.CSV dataset (see FLIP.CMD command file for details):



Battery KEEP

Battery KEEP randomly selects a specified number of variables from the initial list of predictors (controlled by the KEEP command) and repeats the random selection multiple times. A user has the option of specifying the CORE subset of variables that are always present in each run.

We illustrate this battery on the SPAMBASE.CSV dataset by sampling 10 predictors at a time and repeating this process thirty times while requiring the CHAR_FREQ_EXPLAM, and CHAR_FREQ_DOLLAR variables to be always present (see KEEP.CMD command file for details).



The resulting models have an average area under the ROC curve ranging from 87.84% to 93.47%.

The largest ROC model has the following variable importance list:

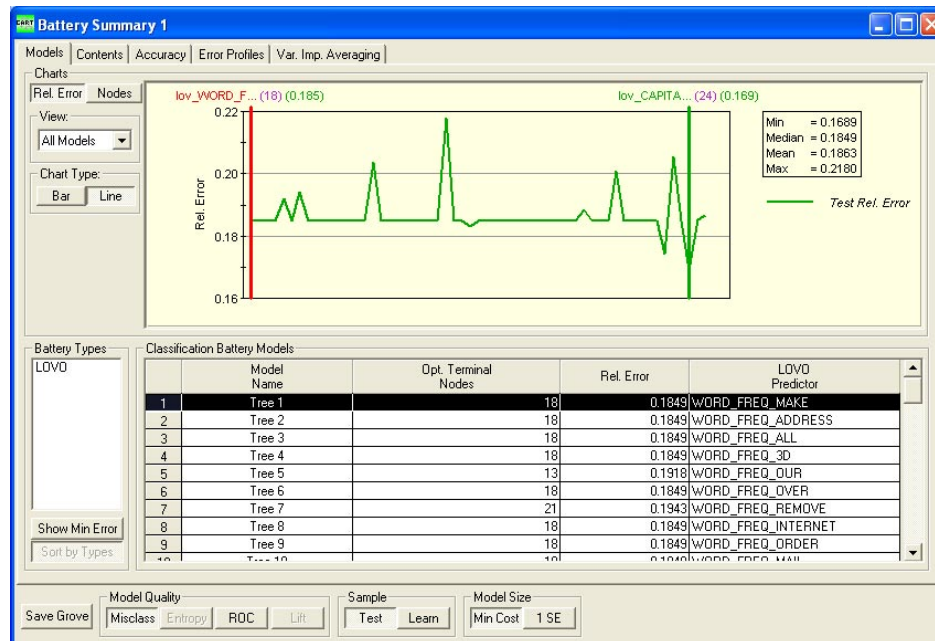
Variable Importance		
Variable	Score	
CHAR_FREQ_DOLLAR	100.00	
CHAR_FREQ_EXPLAM	76.10	
WORD_FREQ_FREE	72.74	
WORD_FREQ_HP	68.68	
WORD_FREQ_OUR	54.12	
WORD_FREQ_650	27.80	
WORD_FREQ_85	13.19	
WORD_FREQ_INTERNET	10.54	
WORD_FREQ_PM	5.23	
WORD_FREQ_PEOPLE	1.46	



Battery LOVO

Battery LOVO (Lease One Variable Out) generates a sequence of runs where each run omits one of the variables on the predictor list one at a time. Assuming K predictors on the initial keep list, the battery produces K models having K-1 predictors each.

We illustrate this battery on the SPAMBASE.CSV data using the full list of predictors (see LOVO.CMD command file for details).



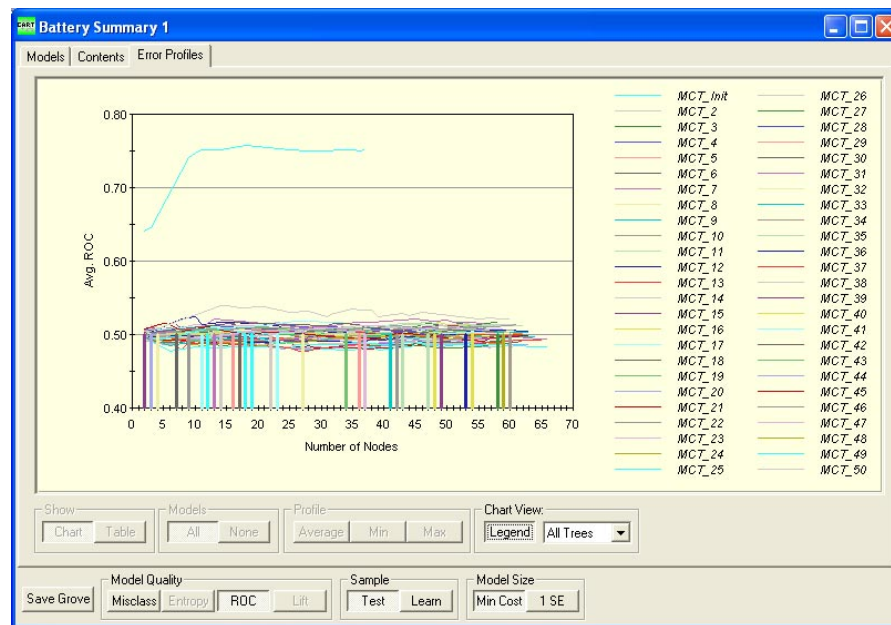
According to the results, removing CAPITAL_RUN_LENGTH_AVERAGE from the predictor list actually improves the relative error to 0.169.



Battery MCT

Battery MCT generates a Monte Carlo test on the significance of the model performance obtained in a given run. The target is first randomly permuted (thus destroying any possible dependency of the target on all remaining variables), and then a regular model is built. The process is repeated many times and the resulting profiles are shown together with the actual run profile. One would want to see the actual run profile as far away from the MCT profiles as possible.

We illustrate this battery on the SPAMBASE.CSV data using a small list of predictors (see MCT.CMD command file for details).



It is clear that even this arbitrarily chosen set of predictors is capable of capturing some useful signal. Note that the family of MCT profiles results in a test ROC within 48% and 54%. It would have been difficult to justify the legitimacy of a model having a ROC value within this region.



Battery MINCHILD

Battery MINCHILD is very similar to battery ATOM described above. It varies the required terminal node size according to a user-supplied setting.



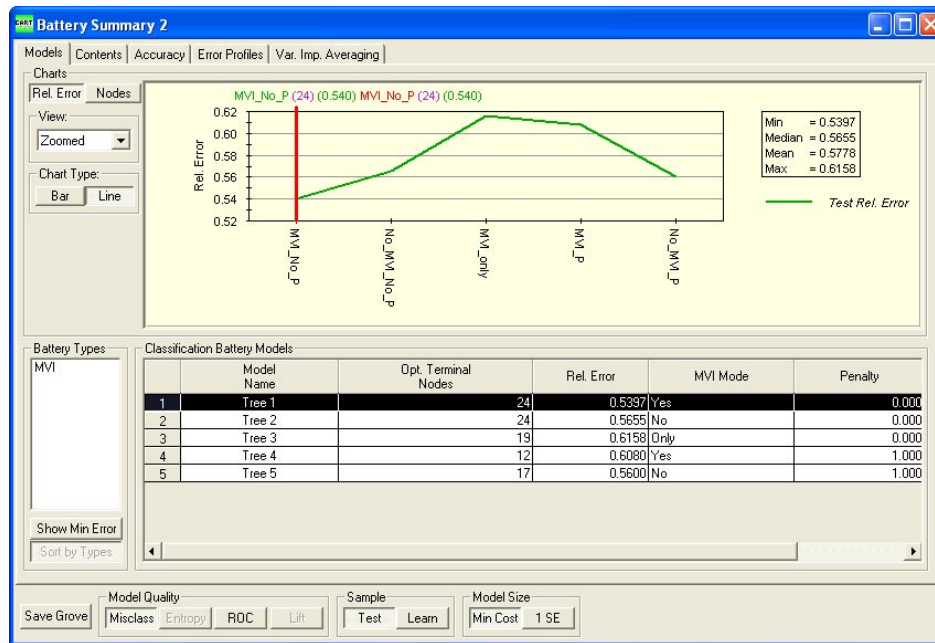
Battery MVI

Battery MVI addresses missing value handling, which is important for the success of any data mining project. CART has a built-in default ability to handle missing values via the mechanism of surrogate splits (alternative rules automatically invoked whenever the main splitter is missing). Surrogate splits effectively redistribute the missing part of data between the left and right sides of the tree based on an alternative split that most resembles the local split. This is fundamentally different from treating a missing value as a separate category, thus sending the entire subset to one side.

Alternatively, it is often important to find out whether the fact that one variable is missing can be predictive on its own. In CART this can be accomplished by creating missing value indicator variables (MVIs – binary variables set to one when the variable of interest is missing and zero otherwise) and subsequently using the MVIs as part of the analysis (see the **Model Setup—Advanced** tab).

In addition, CART allows variables that have missing values to be penalized. The amount of penalty is usually proportional to the percent missingness, thus discouraging variables with heavy missingness from becoming part of the model (**Model Setup—Penalty** tab).

This proliferation of controls over missing value handling in CART essentially leads us to support a whole new kind of battery—battery MVI. Currently, the battery offers a series of five runs with the most interesting combinations of missing value settings. We illustrate this battery using FNCELLA.CSV, the Cell Phone dataset (see MVI.CMD command file for details):



The following five models are defined:

- ◆ **MVI_No_P** – use regular predictors, missing value indicators, and no missing value penalties
- ◆ **No_MVI_No_P** – use regular predictors only (default CART model, no MVIs, no penalties)
- ◆ **MVI_only** – use missing value indicators only (no regular predictors, no penalties)
- ◆ **MVI_P** – use regular predictors, missing value indicators, and missing value penalties
- ◆ **No_MVI_P** – use regular predictors and missing value penalties (no MVIs)

As the graph above indicates, one could reduce the relative error to 0.616 using missing value indicators alone. Such remarkable predictability often indicates meaningful patterns of missing values in the data.



Battery NODES

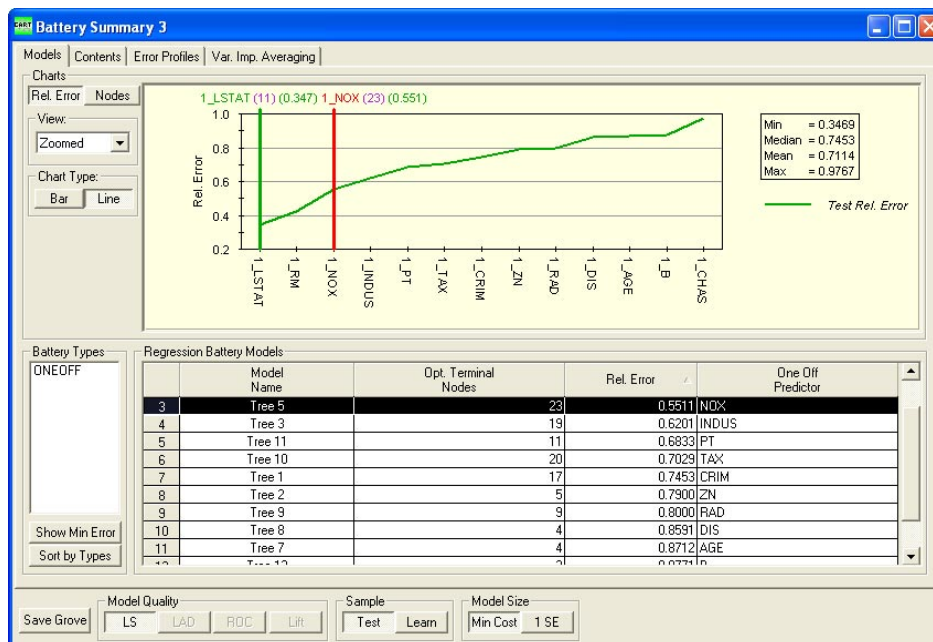
Battery NODES is very similar to battery DEPTH described above. It varies the limit on the tree size in nodes according to a user-supplied setting.



Battery ONEOFF

Battery ONEOFF was designed to generalize conventional co-relational analysis by placing the CART engine in its core. The battery contains the results of using one variable at a time to predict the response.

We illustrate this battery using the BOSTON.CSV dataset (see the ONEOFF.CMD command file for details):



It is clear that LSTAT alone could reduce the relative error to 0.35 while CHAS has virtually no univariate connection with the response.

The following table reports Pearson correlations of the same variables with the response sorted by the absolute value of the correlation. The results are directly

comparable to CART findings. However, the CART approach has the added advantage of being able to identify potential non-linearities.

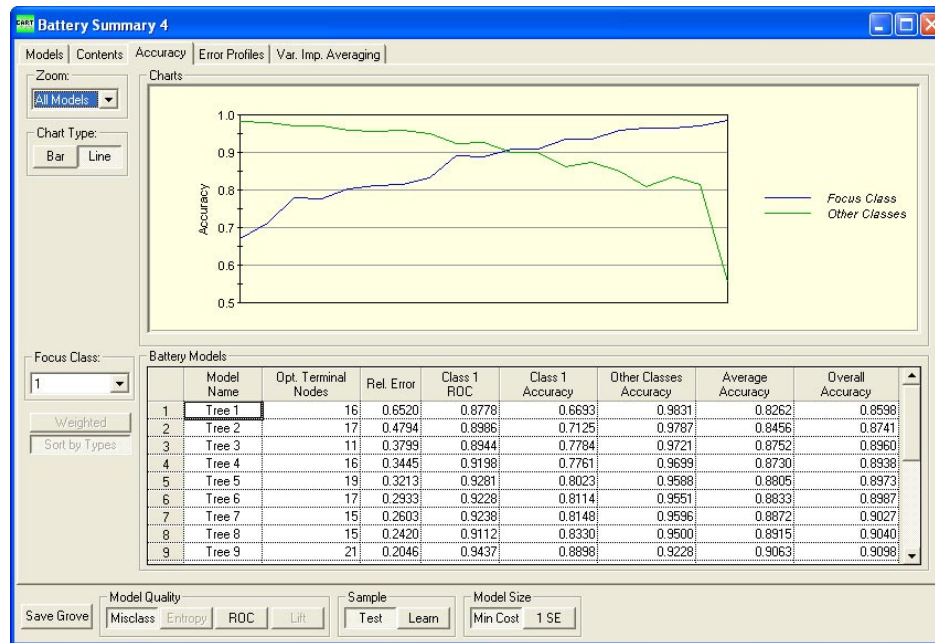
VARIABLE	CORRELATION
LSTAT	-0.73766
RM	0.69536
PT	-0.50779
INDUS	-0.48373
TAX	-0.46854
NOX	-0.42732
CRIM	-0.3883
RAD	-0.38163
AGE	-0.37695
ZN	0.360445
B	0.333461
DIS	0.249929
CHAS	0.17526



Battery *PRIOR*

Prior probabilities play a fundamentally important role in overall tree construction as well as in model evaluation. By manipulating priors one could impose different solutions on the sensitivity versus specificity tradeoff as well as control node purity and overall model performance. Battery PRIOR streamlines this process by allowing priors to be varied within the specified range in user-supplied increments.

We illustrate this battery using the SPAMBASE.CSV dataset (see PRIORS.CMD command file for details):



Here the priors were varied from (0.05, 0.95) to (0.95, 0.05) in increments of 0.05, producing 19 runs overall. Note the powerful impact on individual class accuracies (sensitivity versus specificity tradeoff).

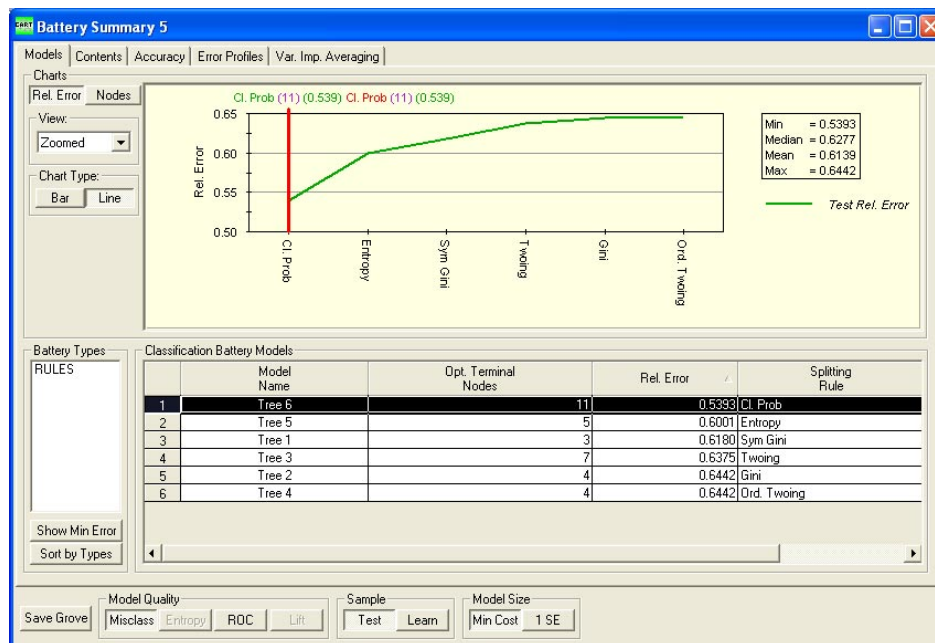
This battery is the most suitable raw material for the hot-spot detection procedure (searching for rich nodes in the class of interest) described earlier.



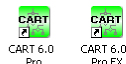
Battery RULES

Battery RULES simply runs each available splitting rule, thus producing six runs for classification and two runs for regression.

We illustrate battery RULES for a multinomial target with non-symmetric costs using the Prostate dataset PROSTATE2.CSV (see RULES.CMD command file for details):



It appears that the Class Probability splitting rule resulted in the smallest relative error while GINI and Ordered Twoing resulted in the largest relative error.

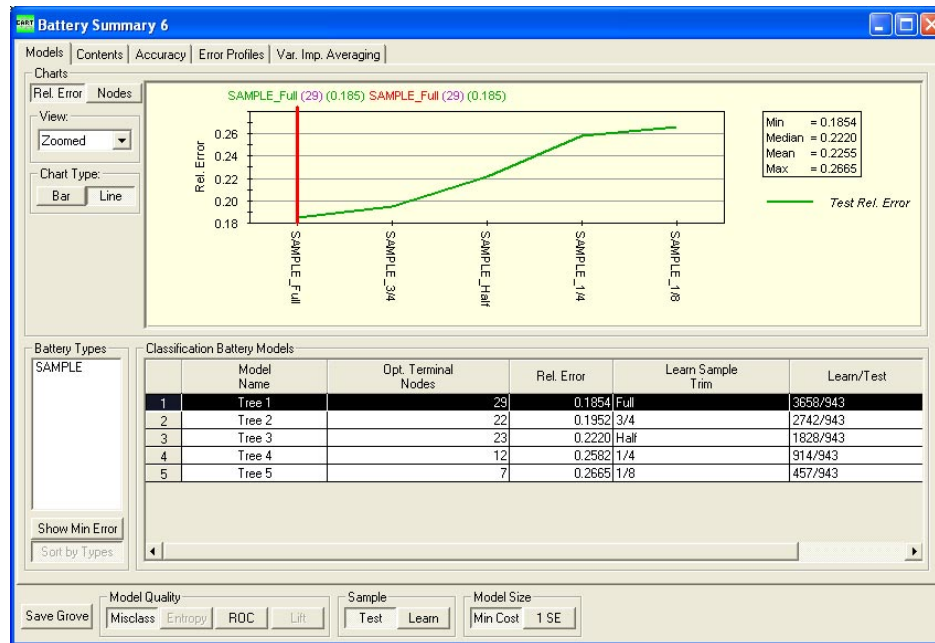


Battery SAMPLE

The CART process iteratively partitions the train data until no more sensible splits can be found. When the train data size is limited, it is possible to run out of support for subsequent splits before the useful signal is fully extracted. CART is sensitive to the overall size of the train data.

Battery SAMPLE was designed to investigate the amount of accuracy loss incurred in the course of progressive reduction of the train data size (observation-wise). A total of five runs are produced: full train data, $\frac{3}{4}$ of the train data, $\frac{1}{2}$ of the train data, $\frac{1}{4}$ of the train data, and $\frac{1}{8}$ of the train data.

We illustrate this battery using the SPAMBASE.CSV data with 20% randomly allocated for test partition (see SAMPLE.CMD command file for details):



Apparently, minor accuracy loss occurs when going from the full sample to $\frac{3}{4}$ of the data. However, the loss becomes substantial when $\frac{1}{2}$ or more of the data are eliminated.



Battery SHAVING

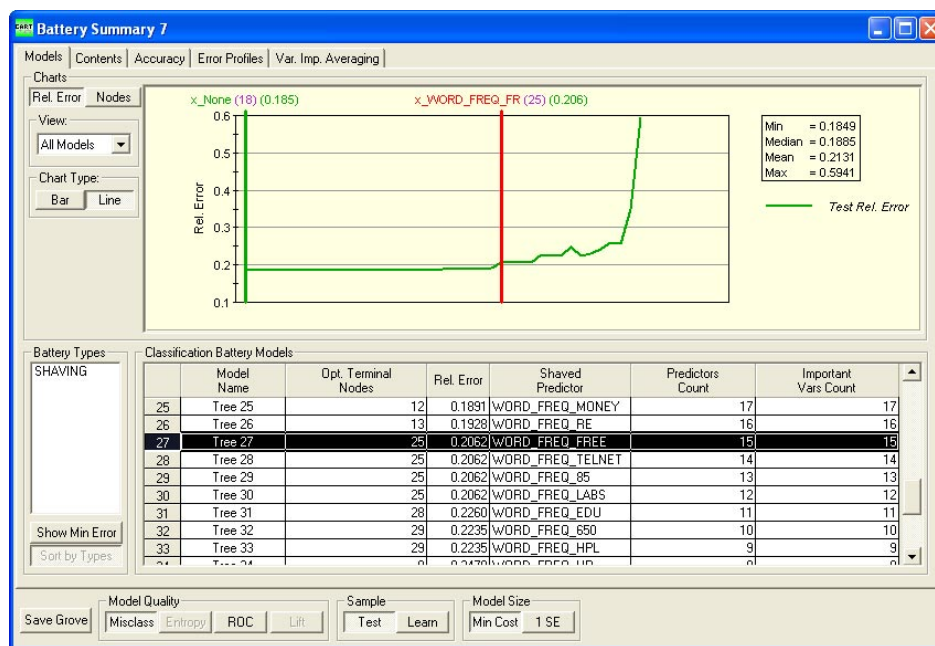
Battery SHAVING was inspired by conventional step-wise regression modeling techniques. The key idea is to build a model, study the reported variable importance, and proceed by eliminating one or a group of variables based on a specified strategy.

The following shaving strategies are currently available (assuming K starting variables):

- ◆ **BOTTOM** – remove the least important variables (up to K runs)
- ◆ **TOP** – remove the most important variables (up to K runs)
- ◆ **ERROR** – remove the variable with the least contribution based on the LOVO battery (see above) applied to the current set of variables (up to $K(K-1)/2$ runs)

By default, each battery starts with the current list of predictors and proceeds until no predictors are left. The user can change both the number of steps (elimination cycles) taken and the number of variables removed at each step (one by default).

We illustrate this process by shaving from the bottom of the entire initial list of predictors in the SPAMBASE.CSV data (see SHAVING.CMD command file for details):



It follows that the original list of 41 important predictors can be reduced to only 15 predictors without substantial loss of accuracy.



Battery SUBSAMPLE

Battery SUBSAMPLE varies the sample size that is used at each node to determine competitor and surrogate splits. The default settings are no subsampling followed by subsampling of 100, 250, 500, 1000 and 5000. You may list a set of values with the VALUES option as well as a repetition factor. Each subsampling size is repeated N times with a different random seed each time.

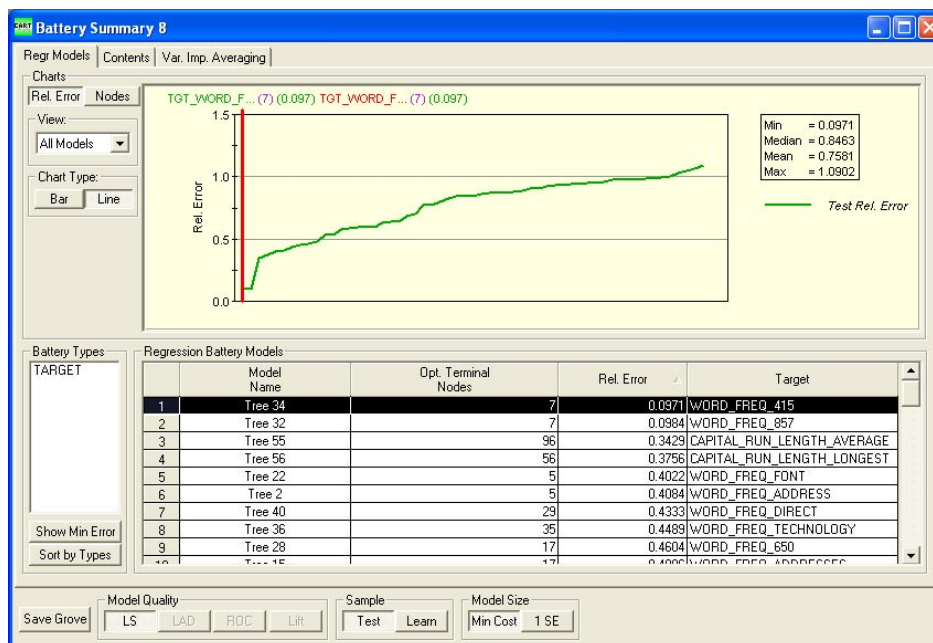


Battery TARGET

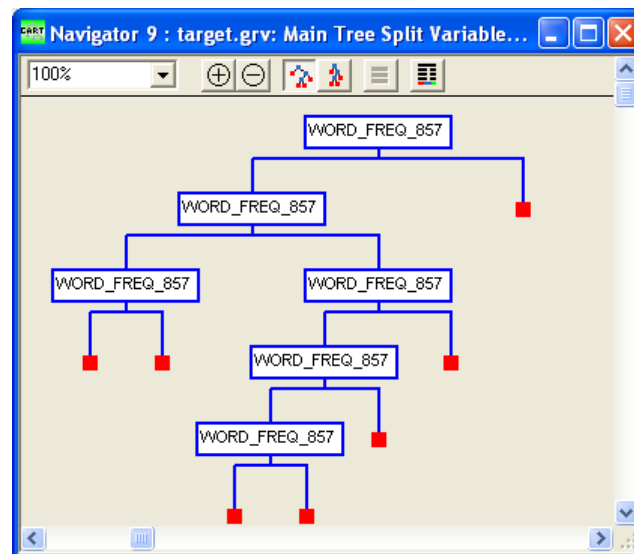
While theoretical research usually assumes independence among predictors, this assumption is almost always violated in practice. Understanding the mutual relationship among a given list of predictors becomes important in a variety of contexts. A traditional covariance matrix may provide insight into pair-wise correlations among predictors, but usually fails to capture any serious multivariate relationships or possible non-linearities.

Battery TARGET was designed to overcome the limitations of conventional approaches and construct a more reliable measure of inter-dependency. The process proceeds as follows: each variable from the current predictor list is taken as a target and a model is built to predict this target (classification tree for categorical predictors and regression tree for continuous predictors) using the remaining variables. The resulting model accuracy indicates the degree of association between the current target and the rest of the variables while the variable importance list tells exactly what variables are involved.

We illustrate this process using the SPAMBASE.CSV dataset (see TARGET.CMD command file for details):



The results indicate that WORD_FREQ_415 is the easiest to predict (relative error 0.0971). Double clicking on the highlighted line and looking at the Splitters information in the resulting navigator reveals:



In other words, WORD_FREQ_857 can be used to predict WORD_FREQ_415 nearly perfectly.

In contrast, WORD_FREQ_PARTS cannot be predicted with any reasonable accuracy at all (relative error 1.09 is greater than 1.0).

CART Segmentation

*A classification/segmentation example to
illustrate the multi-class problem*

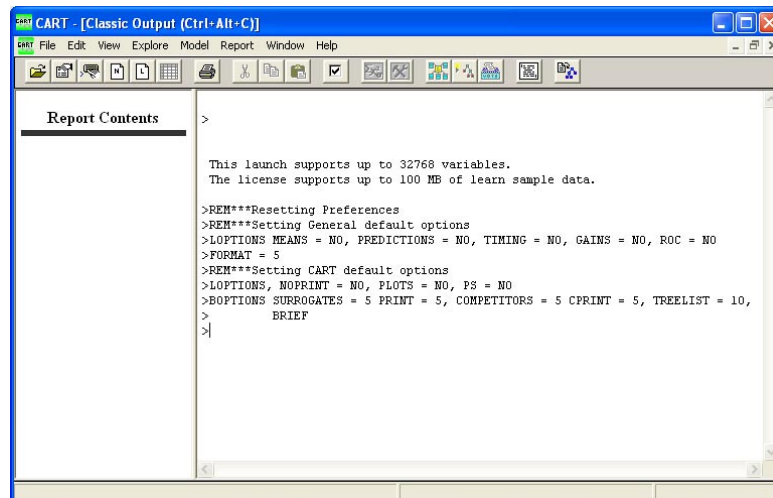
Modeling the multi-class target

So far we have discussed two-class classification examples. In this chapter we walk through a simple three-class example to illustrate some of the unique aspects of this form of modeling. In the example that follows, we analyze a data set containing information on health club members who have been classified into three market segments. The goal of our analysis is to uncover the important factors that differentiate the three segments from each other. The variables in the GYMTUTOR.CSV data set (included on your installation CD) are:

SEGMENT	Member's market segment (coded 1,2,or 3)
ANYRAQT	Racquet ball usage (binary indicator coded 0, 1)
ONAER	Number of on-peak aerobics classes attended
NSUPPS	Number of supplements purchased
OFFAER	Number of off-peak aerobics classes attended
NFAMMEM	Number of family members
TANNING	Number of visits to tanning salon
ANYPOOL	Pool usage (binary indicator coded 0, 1)
SMALLBUS	Small business discount (binary indicator coded 0, 1)
FIT	Fitness score
HOME	Home ownership (binary indicator coded 0, 1)
PERSTRN	Personal trainer (binary indicator coded 0, 1)
CLASSES	Number of classes taken

CART Desktop

Double-click on the CART program icon and you will see the following screen:




About CART Menus

The menu items in CART change depending on the stage of your analysis and which window is actively in the foreground. As a result, some menus may be disabled if not available. Similarly, the commands that appear in the pull-down menus and the toolbar icons are disabled if not accessible. An overview of the layout of the main CART menus is presented below.

- FILE**
 - Open data set, Navigator file, or command file
 - Save analysis results, Navigator file, Grove file, or command file
 - Export tree rules
 - Specify printing parameters
 - Activate interactive command mode
 - Open notepad
 - Submit batch command files
- EDIT**
 - Cut, copy and paste selected text
 - Specify colors and fonts
 - Control reporting options
 - Set random number seed
 - Specify default directories
- VIEW**
 - Open command log
 - View data
 - View descriptive statistics
 - Display next pruning
 - Assign class names and apply colors
 - View main tree and/or sub-tree rules
 - Overlay gains charts
 - Specify level of detail displayed in tree nodes
- EXPLORE**
 - Generate frequency distributions
- MODEL**
 - Specify model setup parameters
 - Grow trees/committee of experts
 - Generate predictions/score data
 - Translate models into SAS[®], C, or PMML
- TREE**
 - Prune/grow tree one level
 - View optimal/minimum cost/maximal tree
 - View tree summary reports
- REPORT**
 - Control CART reporting facility
- WINDOW**
 - Control various windows on the CART desktop
- HELP**
 - Access online help

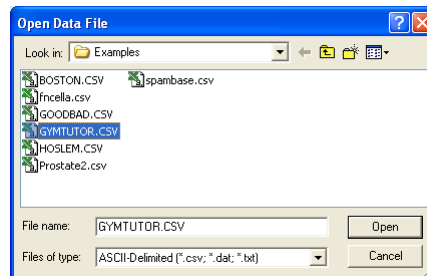
Opening a File

To open the input data file GYMTUTOR.CSV used in our example:

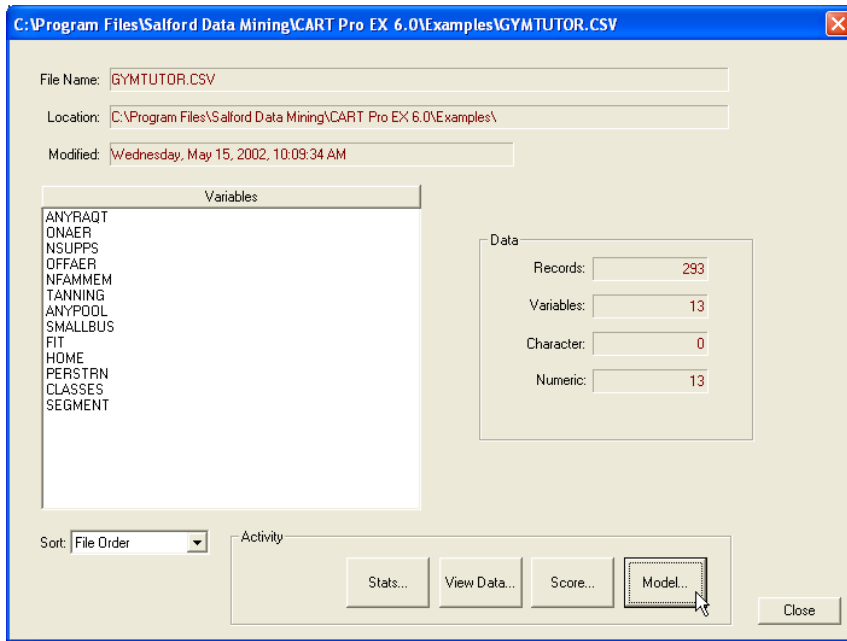
1. Select Open->Data File... from the File menu (or click on the  toolbar icon).

 Note that you can set default input and output directories; select **Options...** from the **Edit** menu and select the **Directories** tab.

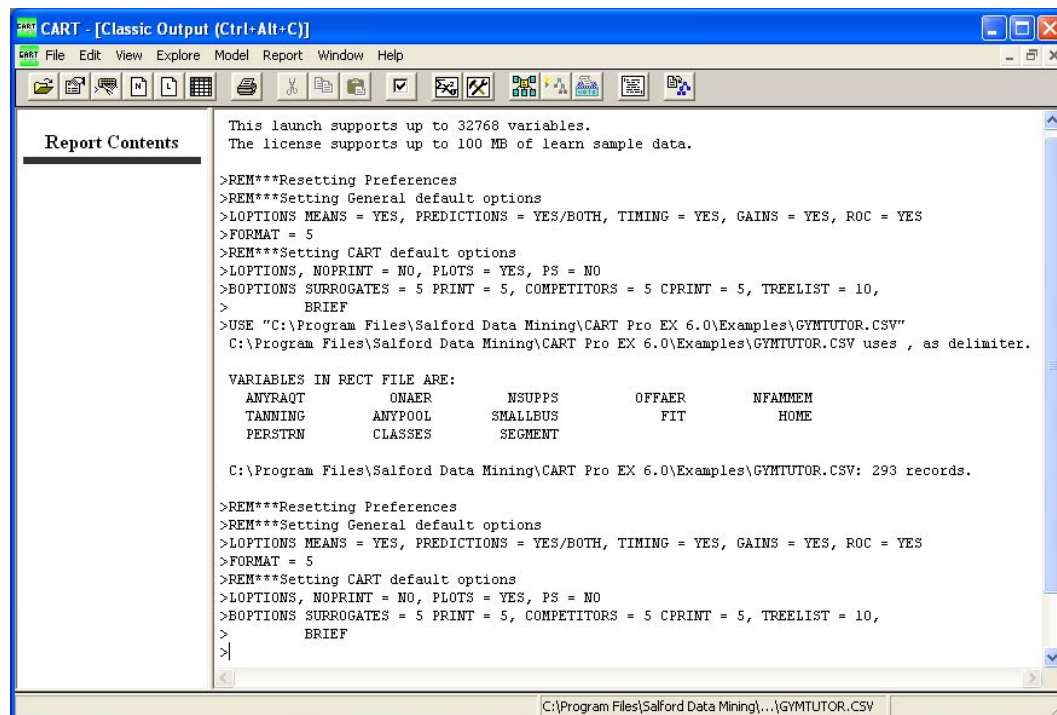
In the **Open Data File** dialog, select the GYMTUTOR.CSV file from the Sample Data folder and click on **[Open]** or double-click the file name. (As indicated below, **Delimited Text (*.csv, *.dat, *.txt)** must be selected in the **Files of Type:** box to see files ending with the .CSV extension.)



After you open GYMTUTOR, a dialog opens automatically that gives information on the dataset and allows one to choose between data viewing, stats, modeling or scoring.



If the Model button is clicked on, the **Model Setup** dialog opens and the **CART Output** window appears in the background. Hyperlinked **Report Contents** appears in the left panel of the Output window and text output in the right. The initial text output contains the variable names, the size of the file, and the number of records read in.



Setting Up the Model

The Model Setup dialog tabs are the primary controls for conducting CART analyses, with the analysis functions most commonly used conveniently located in eleven Model Setup tabs. After you open a data set, setting up a CART analysis entails several logical groups, all of which are carried out in one of the Model Setup dialog tabs.

Model	select target and predictor variables, specify categorical predictors and weight variables, choose tree type (classification, regression, or cluster analysis).
Categorical	set up categorical class names.
Force Split	specify a split variable for the root node and its immediate children.
Constraints	allows you to pre-specify sets of variables to be used in specific regions of the tree and to determine the order in which splitters appear in the tree. This is a CART ProEX feature only.
Testing	select a testing or self-validation method.
Select Cases	select a subset of original data.
Best Tree	define the best tree selection method.
Method	selecting a splitting rule.
Advanced	specify other model-building options.

Cost	specify misclassification costs.
Priors	specify priors.
Penalty	set penalties on variables, missing values, and high-level categorical predictors.
Battery	specify a battery of models to be run.

The only **required** step is the first one: specify a target variable and tree type in the Model Setup dialog.

In our tutorial example, we enter information into the **Model** tab only and then grow the tree using CART's default settings: cross validation, Gini splitting rule, unitary misclassification costs, and equal priors. When the other Model Setup dialog tabs are left unchanged, the following defaults are used:

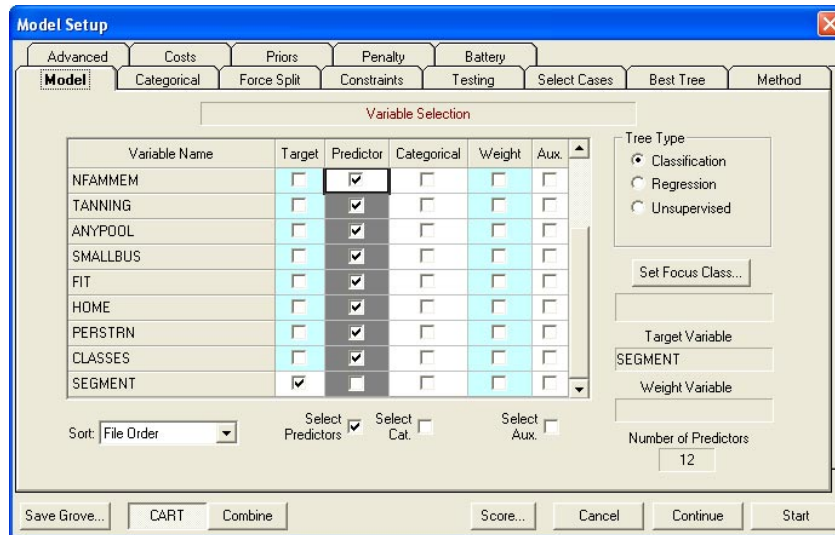
- ◆ All remaining variables in the data set other than the target will be used as predictors (the **Model** tab)
- ◆ No weights will be applied (the **Model** tab)
- ◆ 10-fold cross validation for testing (the **Testing** tab)
- ◆ Minimum cost tree will become the best tree (the **Best Tree** tab)
- ◆ Only five surrogates will be tracked and will count equally in the variable importance formula (the **Best Tree** tab)
- ◆ GINI splitting criterion for classification trees and least squares for regression trees (the **Method** tab)
- ◆ Unitary (equal) misclassification costs (the **Costs** tab)
- ◆ Equal priors (the **Priors** tab)
- ◆ No penalties (the **Penalty** tab)
- ◆ Parent node requirements set to 10 and child node requirements set to 1 (the **Advanced** tab)
- ◆ Allowed sample size set to the currently-open data set size (the **Advanced** tab)
- ◆ 3000 limit for cross-validation warning

Additional tree-building and -reporting options include enabling linear combination splits, combining trees using bagging or ARCing, limiting the size and structure of the tree, filtering data and limiting the size of the test and learn samples, exporting tree rules, and identifying where to permanently save the CART output, navigator file and tree models. (See Chapters 3, 4, and 5 for further discussion of the default settings and an extended tutorial using the other nine Model Setup dialogs.)

Selecting Target and Predictor Variables

For this analysis, the three-level categorical variable SEGMENT (1,2 or 3) is the target (or dependent) variable. To specify the target variable, use the ▼ to scroll down the variable list until SEGMENT is visible. Put a checkmark inside the checkbox located in the **Target** column.

If no predictor variables are selected using a checkmark in the **Predictors** column, CART by default includes every variable except the target in the analysis—the desired result in this example. The Model tab now appears as follows:



Categorical Predictors

Put checkmarks in the **Categorical** column against those predictors that should be treated as categorical. For our example, specify ANYRAQT, TANNING, ANYPOOL, SMALLBUS, and HOME as categorical predictor variables.

Growing the Tree

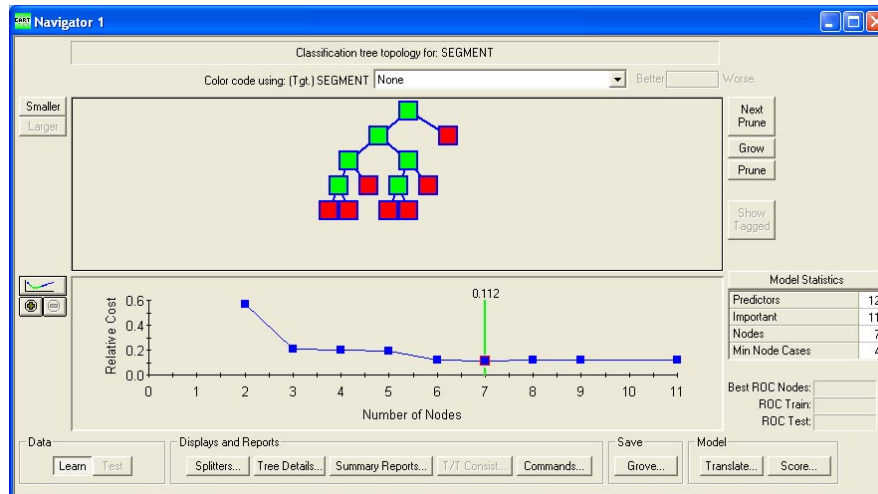
The **Classification** radio button is already selected in the **Tree Type** group box by default. Likewise, the **[CART]** button is depressed, or ON, by default. We are now ready to grow our tree.

To begin the CART analysis, click the **[Start]** button. A progress report appears that lets you know how much time the analysis should take and approximately how much time remains. Once the analysis is complete, text output appears in the **CART**

Output window, blue hyperlinks appear in the **Report Contents** panel, and a new window, the **Navigator**, is opened and placed in the foreground. We first explore the Navigator and then return to the text output.

Tree Navigator

The tree topology, displayed in the top panel of the Navigator window, provides an immediate snapshot of the tree's size and depth. By default, the optimal or minimum cost tree is initially displayed and, in this example, is the tree with nine terminal nodes, as illustrated below.

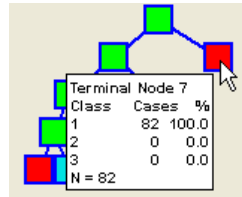


Terminal nodes in classification trees are color coded to indicate whether a particular class level improves or worsens with respect to terminal node purity when compared to the root node. By default, color-coding is not initially displayed. To activate color-coding, select a target class level "1" from the control next to the **Color code using...** title box located at the top of the Navigator.

As the legend in the upper-right corner of the Navigator indicates, nodes better than the root node are shades of red, while nodes worse than the root node are shades of blue. The more saturated the color, the greater the improvement or worsening in that terminal node when compared to the root node.

CART terminal nodes are numbered left to right in ascending order, starting with one. In our example, we can quickly ascertain that Class 1 cases are concentrated primarily in red terminal nodes 1, 3, and 7, whereas very few or no Class 1 cases populate the remaining blue terminal nodes.

Hovering the mouse over any of the nodes results in additional node information popping up.



✎ You may change how many details are shown by a right-mouse click in the “gray” area of the navigator window and a further left-mouse click on the node information sample display. Alternatively, one may do the same using the **View->Node Display** menu.


The bottom panel of the Navigator provides a visual indication of the quality of the optimal tree—a graph displaying the cross-validated relative cost by the number of terminal nodes for each tree size in the nested tree sequence. Recall that CART begins by growing the maximal or largest tree possible and then prunes those sections of the tree that contribute least to overall accuracy, pruning all the way back to the root node. As we would expect, the relative cost, or misclassification rate, goes down as the tree gets progressively larger but, at a certain point, plateaus (and, in some cases, will begin to climb).

CART's Navigator allows you not only to explore the different tree topologies, but to interactively inspect detailed summary and diagnostic information for each “sub-tree” in the tree sequence. To explore a different tree you may:

- ♦ click on the blue box in the line graph, or
- ♦ from the **T**ree menu, choose **S**elect Tree, or
- ♦ use left and right arrow keys, or
- ♦ click **[G**row] or **[P**run] buttons, or
- ♦ select **G**row One Level or **P**run One Level in the **T**ree menu.

The tree size you select appears in the top panel of the Navigator window. The ability to see different trees is particularly useful if you feel the optimal CART tree is too large or if you are only concerned with the first few splits in the tree.

You can also see the tree nodes that will be pruned next as you move one step down the tree sequence by selecting **Show N**ext **P**runing from the **V**iew menu or pressing the **[N**ext **P**run]]button in the Navigator window; these nodes are outlined in bright yellow.

Repeated pressing of the  button cycles through three alternative displays in the lower half of the Navigator:

- ◆ standard Relative Cost curve
- ◆ color-coded Relative Cost curve
- ◆ percent population by node display

The first two displays show the relative cost curve depending on the number of terminal nodes, while the last display reports how the original data set is distributed into the terminal nodes in the currently-selected tree.

✎ If you click on an individual bar in the “percent population by node” display, the corresponding node in the tree topology becomes yellow.

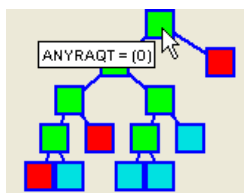
Pressing on the **[Smaller]** or **[Larger]** button causes the scale of the tree topology in the top half of the navigator window to become larger or smaller. This is useful when analyzing very large trees.

When applicable, you may switch between learn or test counts displayed for each node by pressing the **[Learn]** or **[Test]** button. Because cross validation was used in this example, only learn counts are available on the node-by-node basis.

You can also save the Navigator or Grove file (needed for scoring) by pressing the **[Grove...]** button, or you may translate CART models into SAS®, C, PMML, or Java representations by clicking the **[Translate...]** button. Finally, you may apply any tree to data using the Score dialog accessed via the **[Score...]** button. See Chapter 7 for step-by-step instructions for scoring new data.

Viewing Variable Splits

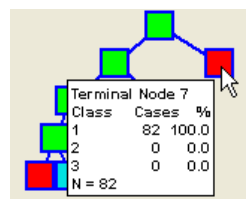
By hovering the mouse pointer over a non-terminal (green) node, you initially see terse information about the split, as illustrated below.



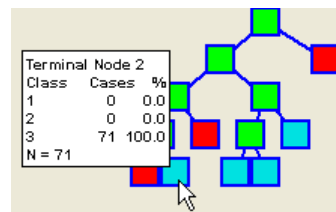
The root node split on the variable ANYRAQT, a binary indicator variable coded 1 if the member uses the racquetball courts and 0 otherwise. Members who do not use the racquetball courts go to the left non-terminal node while those who use the courts go to the right terminal node.

To see more or less detail when hovering over a node, activate a local menu by clicking the right mouse button on the background (or select **Node Display** from the **View** menu) and then select the level of detail you prefer. You can elect to see the splitting variable name, the splitting criterion, the class assignment, the class breakdown (counts and percentages) and the number of cases in the node.

If we select the most detailed node report and hover the mouse pointer over the terminal node on the far right (terminal node 7), we can see a very good split: 82 of the original 95 Class 1 cases and none of the Class 2 or 3 cases appear in this node. Thus, based on the first split only, we already know something about these particular members, that is, if ANYRAQT=1, then SEGMENT=1. Similarly, the terminal node on the far left side (terminal node 2) shows that after four splits, CART is able to separate 71 of the original 98 Class 3 cases into another pure node.



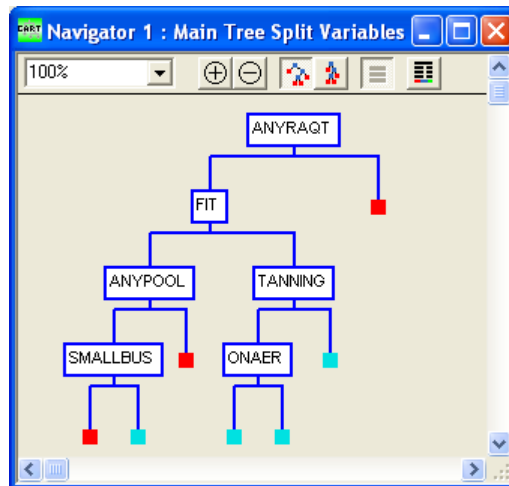
Terminal Node 7



Terminal Node 2

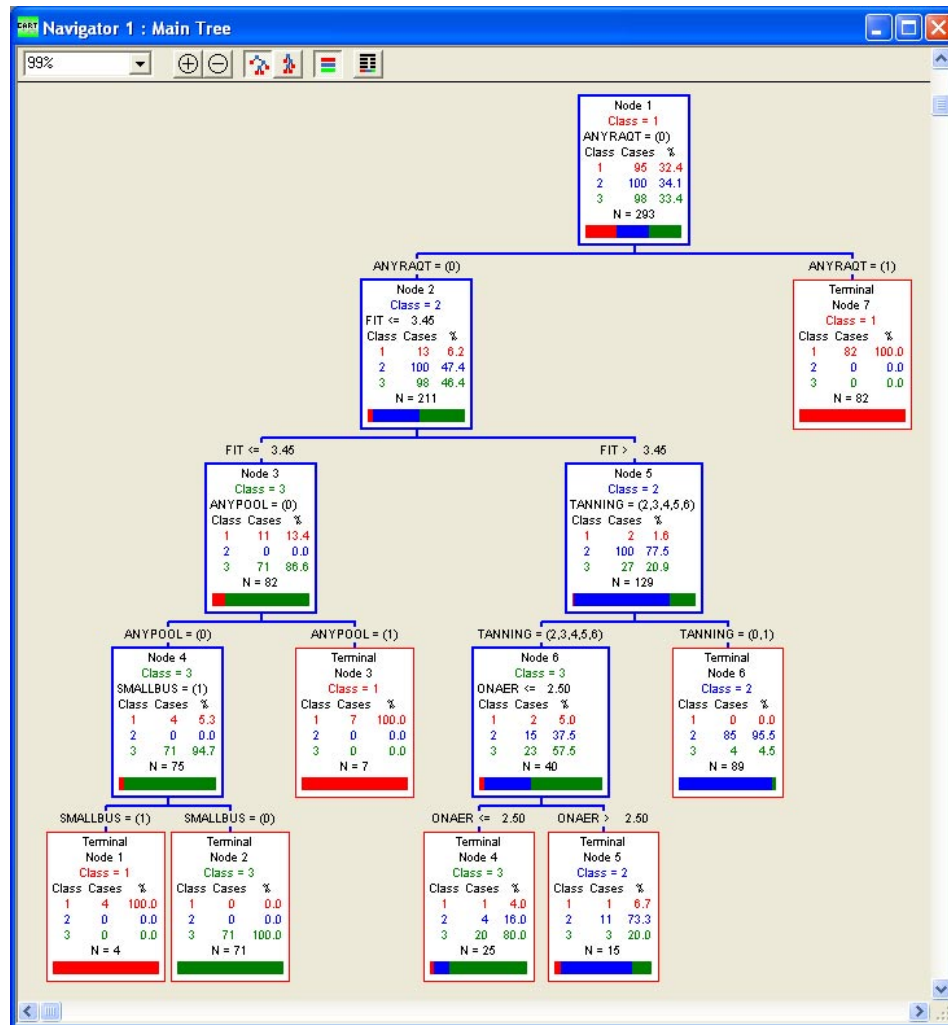
Viewing the Main Splitters

You can quickly scan all main splitters in the entire tree by clicking on the **[Splitters...]** button at the bottom of the Navigator. This display is often useful to quickly identify where in the tree, if at all, a certain variable of interest showed as the main splitter.




Viewing the Main Tree







In addition to the thumbnail sketch of the individual nodes (tree topology) or splitters window, you can view a complete picture of the tree in a format similar to the way the tree will print. To view the entire tree, click on the **[Tree Details...]** button at the bottom of the Navigator (or right-click on the root node and select **Display Tree**).



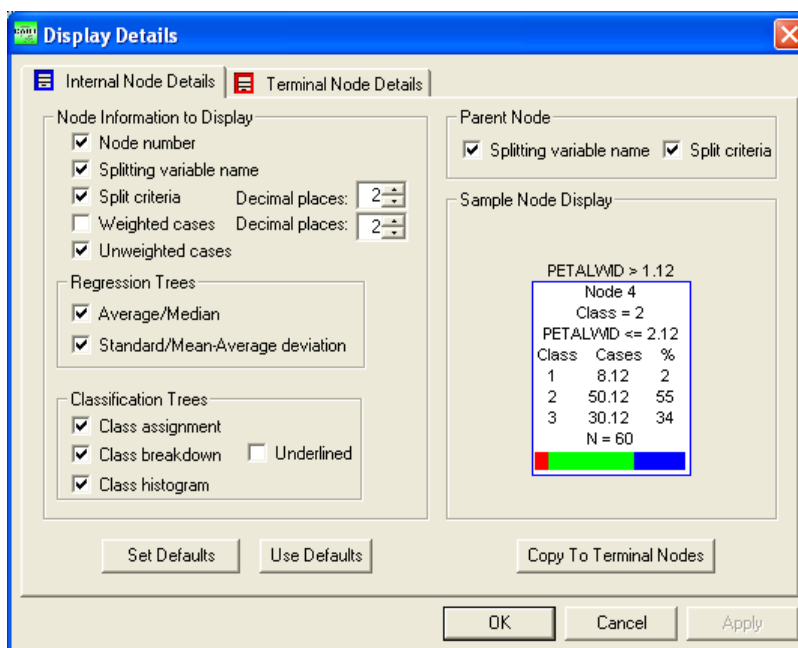
As illustrated above in the upper left, a **Tree Map** window shows a thumbnail sketch of the whole tree and outlines the portion of the tree currently displayed in the Tree window. If the tree takes up more than the screen, you can use the tree map to see which portion of the tree you are viewing and to change the displayed section. Clicking on the tree map moves the viewed portion to center on the mouse position. Conversely, the outline in the map and the section of the tree displayed move when you use the horizontal and vertical scroll bars.

 Tree Map is also available when viewing the Splitters window.

With a simple mouse click you may:

- ◆ Zoom in or Zoom out by pressing the  or  keys
- ◆ Fine-tune the scale by changing the  selection box
- ◆ Experiment with two alternative node-spacing modes ( and  buttons)
- ◆ Turn color coding of target classes on or off ( button)

The level of detail appearing in each of the tree nodes can be customized according to your preferences. From the **View** menu, select **Node Detail...**; the following dialog appears:



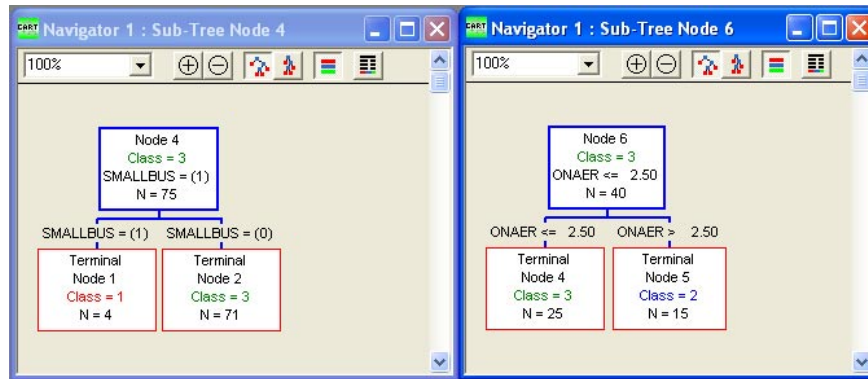
The default display setting is shown in a sample node in the right panel. Click on the check boxes to turn each option on and off and then click **[OK]** to update the Main Tree display. To save your preferred display options as the default settings, click the **[Set Defaults]** button.

Also note that you may separately control the display of internal nodes versus terminal nodes. Press the **[Copy to Terminal Nodes]** or **[Copy to Internal Nodes]** button if you wish the current setup to be copied into the other tab.

- The **[Set Defaults]** button only sets the defaults for the current tab. If you want to set defaults for both terminal and internal nodes, press this button twice, once for each tab.

Viewing Sub-trees

You can also view sub-trees, different sections of the tree, by right-clicking on an internal node that originates the branch you want displayed and selecting **Display Tree**. As with the main tree, the level of node detail can be changed by selecting **Node Detail...** from the **View** menu. As illustrated below, separate sections of the tree can be displayed side by side by opening a second sub-tree window (the two windows are automatically positioned side by side).

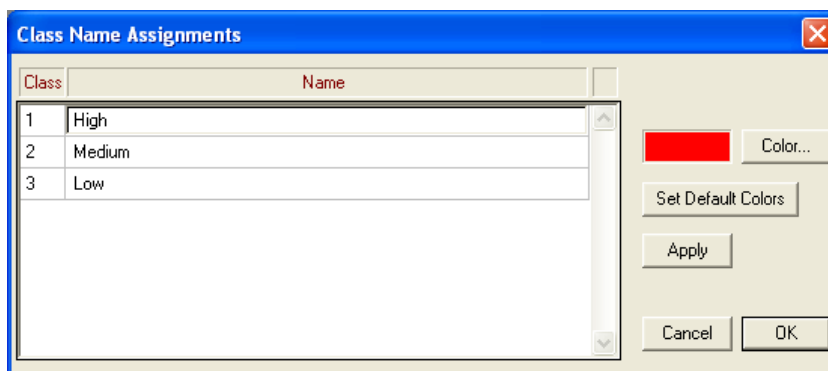


Assigning Labels and Color Codes

Class names (32-character maximum) and colors can also be assigned to each level of the target variable:

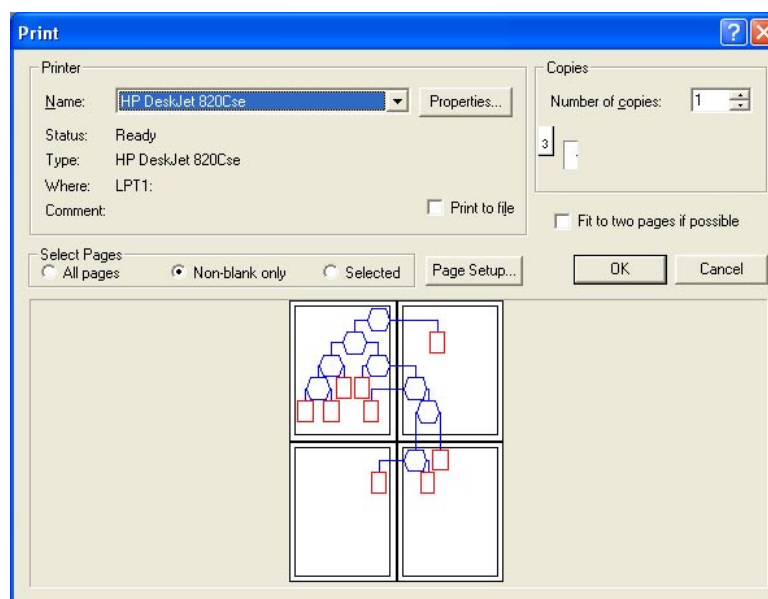
4. Select **Assign Class Names...** from the **View** menu.
5. Click on the **Name** text box and enter a label for that class.
6. Click on **[Color...]**, select a color from the palette, and click **[OK]**.
7. Click **[Apply]** to enter the name/color; repeat steps 2-4 for the other levels.

An illustrative Class Assignment dialog box for our example is shown below. The labels and color codes are displayed in the individual node detail you see when you hover the mouse pointer over a node in the **Navigator** window, as well as in the main and sub-tree diagrams and printed tree output.



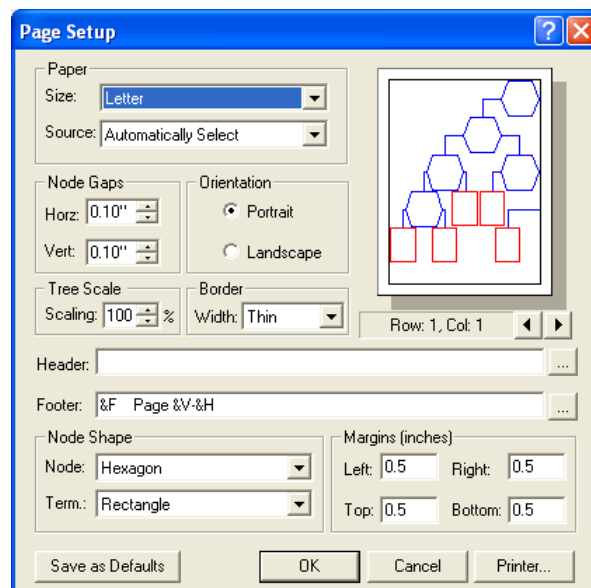
Printing the Main Tree

To print the Main Tree, bring the tree window to the foreground and then select **Print** from the **File** menu (or use **<Ctrl+P>**). In the Print dialog box, illustrated below, you can select the pages that will be printed and the number of copies, as well as specify printer properties. You can also preview the page layout; CART will automatically shift the positions of the nodes so they are not split by page breaks.



You can see from the preview that a small section of the GYMTUTOR main tree spills over to a second page. To resize the tree to fit on one page, click on the **[Page Setup...]**.

The current layout is depicted in the tree preview window of the **Page Setup** dialog shown below. As you change the settings, the print-preview image changes accordingly. To change which page is previewed, use the left and right arrows just below the sample page image.

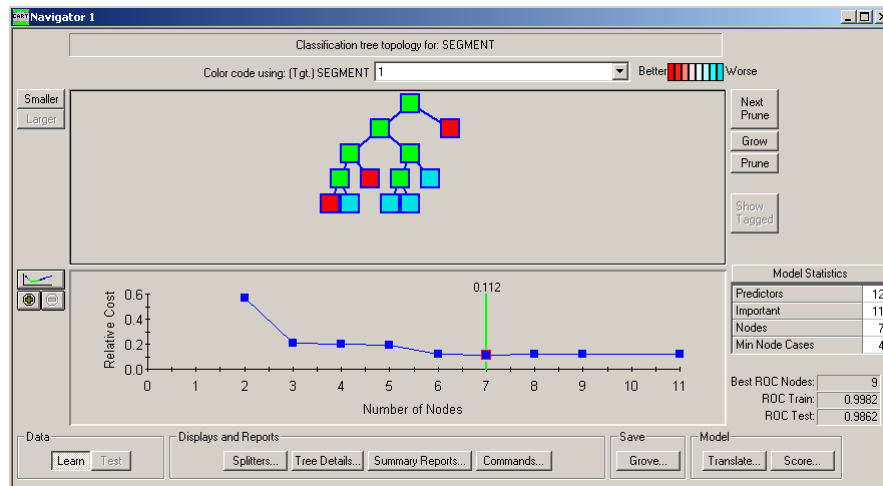


In our example, changing the orientation to landscape and scaling the tree down to 75% of its original size repositions the tree to fit entirely on one page. Click **[OK]** to return to the Print dialog box and then click **[OK]** to send the tree to the printer. (See Chapter 4 for a description of other page setup options.)

Tree Summary Reports

The overall performance of the current tree is summarized in the five **Summary Reports** dialog tabs. To access the reports, click **[Summary Reports...]** at the bottom of the **Navigator** window (or select **Tree Summary Reports...** from the **Tree** menu).

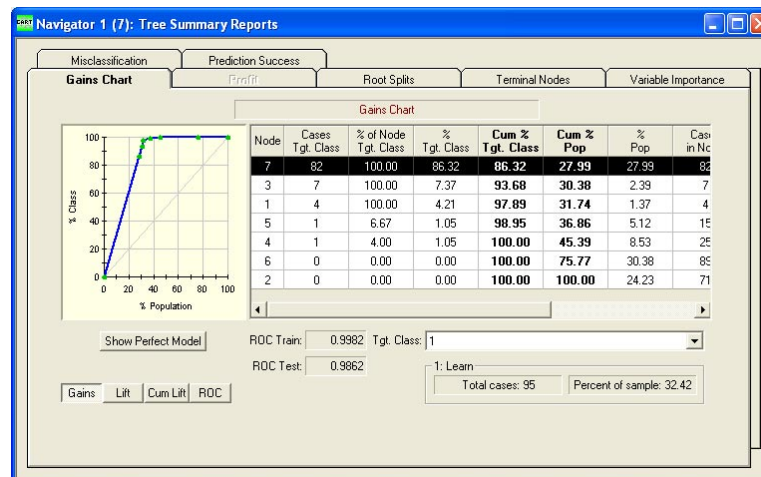
The Tree Summary Reports present information on the currently-selected tree, i.e., the tree displayed in the top panel of the Navigator. To view summary reports for another tree in the nested sequence, change the tree topology displayed in the top panel by selecting the tree of interest (click the square box above the number of nodes on the line graph). Alternatively, you can click on the **Grow** or **Prune** button, or choose **Select Tree** from the **Tree** menu.



As illustrated below, the **Summary Reports** dialog contains gains charts, terminal node distributions, variable importance measures, misclassification tables, and prediction success tables result tabs.

Gains Chart

The summary report initially displayed is the **Gains Chart** tab for the first level of the target variable, Class 1.



If you use a test sample, **[Learn]**, **[Test]**, and **[Both]** buttons will appear in the lower portion of the Gains Chart dialog. To view gains charts for the test sample, click

[Test]. To view gains charts for learn and test combined, click **[Both]**. In this example we used cross validation so these buttons do not appear.

The grid displayed in the right panel shows the relative contribution of the nodes to coverage of a particular class (in this case, Class 1). The nodes are ordered from the richest (highest percentage of Class 1 cases) to the poorest (lowest percentage of Class 1 cases) on the learn data. The table displays the following information for each terminal node (scroll the grid to view the last two columns):

Node	Node reference number
Cases Tgt Class	Number of cases in the node belonging to the target class
% Of Node Tgt Class	Percentage of cases in the node belonging to the target class
% Tgt Class	Number of target class cases in the node as a percentage of the total number of target class cases
Cum. %Tgt Cass	Cumulative number of target class cases as a percentage of the total number of target class cases
Cum. % Pop	Cumulative number of cases as a percentage of the total number of cases in the analysis
% Pop	Percentage of the total number of cases in the analysis that are contained in the node
Cases In Node	Total number of cases in the node
Cum. Gains	Cumulative percentage of target class cases divided by the cumulative share of the number of total cases
Lift Index	Percentage of target class cases in the node divided by percentage of the total number of cases in the node

In the figure displayed in the left panel, the x-axis represents the percentage of the data included and the y-axis represents the percentage of that class included. The 45-degree line maps the percentage of the particular class you would expect if each node were a random sample of the population. The blue curved line represents the cumulative percentage of Class 1 cases (column five in the grid) versus the cumulative percentage of the total population (column six), with the data ordered from the richest to the poorest nodes.

The vertical difference between these two lines depicts the gain at each point along the x-axis. For example, if you use the CART tree to find Class 1 observations and decide to target 30 percent of the population, you would find 91 percent of the Class 1 observations. If you target randomly, you would expect to find only 30 percent of the Class 1 observations. Therefore, the gain in this case is 61 percent (91-30) at x equal to 30. Alternatively, we can say that the lift in this case is $91/30 = 3.03$.

The Gains Table can be exported to Excel by a right-mouse click and then choosing **Export...** from the pop-up menu.

You can print individual Gains Charts as well as overlay and print Gains Charts for trees of different sizes and from different CART analyses (see Chapter 4). You can also add Gains Charts and Tables into the CART report (see Chapter 12).

Root Splits

The next summary report shows the competing root node splits in reverse order of improvement.

Navigator 1 (7): Tree Summary Reports

Misclassification Prediction Success

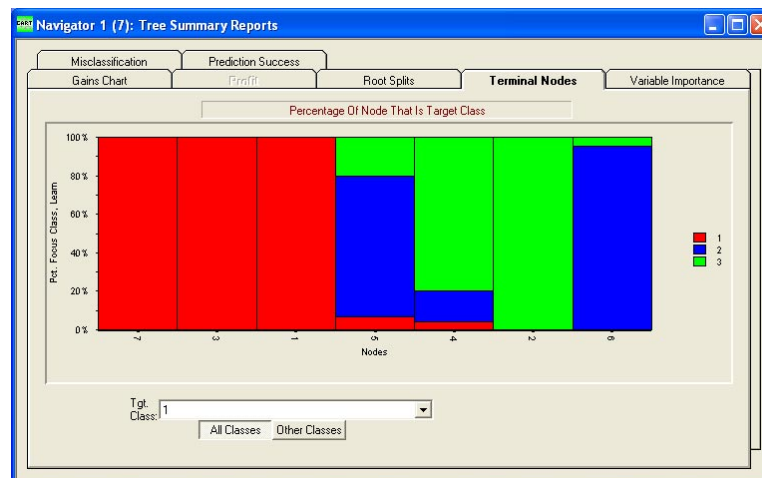
Gains Chart Profit Root Splits Terminal Nodes Variable Importance

Is ANYRAQT = 0

Competitor	Split	Improvement	N Left	N Right	N Missing
Main ANYRAQT	0	0.26929	211	82	0
1 CLASSES	0.50000	0.23363	164	129	0
2 FIT	3.45388	0.23363	164	129	0
3 ANYPOOL	0	0.15684	239	54	0
4 OFFAER	0.50000	0.07206	128	165	0
5 TANNING	0.15	0.06622	166	127	0
6 ONAER	5.50000	0.05951	240	53	0
7 SMALLBUS	1	0.03704	15	278	0
8 NFAMMEM	1.50000	0.02424	10	283	0
9 NSUPPS	1.50000	0.02248	25	268	0
10 HOME	1	0.00579	11	282	0

Terminal Nodes

The next Summary Report provides a graphical representation of the terminal nodes, as illustrated below. You may choose the target class in the selection box. When the **[Other Classes]** button is pressed, the bar chart contains one bar per terminal node sorted by the node richness in the target class. In the example below, terminal nodes 7, 3, and 1 are nearly pure in class 1, whereas only about 5% of node 5 belongs to class 1.



When the **[All Classes]** button is pressed, you will see a stacked bar chart with the target class first.

If you use a test sample, more buttons will be available to reflect distributions on learn, test, or both parts.

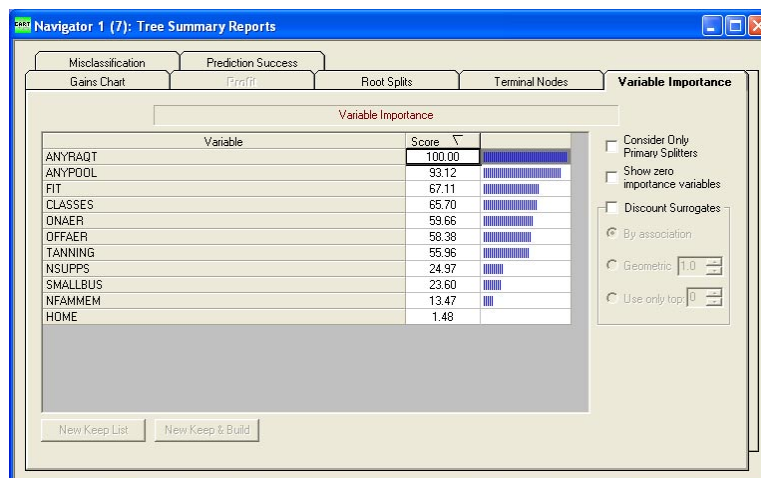
The bar charts enable you to evaluate the purity or homogeneity of the terminal nodes, an indication of how well CART partitioned the classes. The terminal nodes in our example appear to be relatively pure, with six of the nine nodes containing only one class. You can also see how populated each terminal node is and whether particular classes are concentrated in a few nodes or scattered across many nodes, an indication of the number of splits required to partition each of the classes.

Variable Importance

The next Summary Report displays the variable importance rankings, as illustrated below. The scores reflect the contribution each variable makes in classifying or predicting the target variable, with the contribution stemming from both the variable's role as a primary splitter and its role as a surrogate to any of the primary splitters. In our example, ANYRAQT, the variable used to split the root node, is ranked as most important. PERSTRN received a zero score, indicating that this variable played no role in the analysis, either as a primary splitter or as a surrogate.

To see how the scores change if each variable's role as only a primary splitter is considered, click the **Consider Only Primary Splitters** check box; CART automatically recalculates the scores. You can also discount surrogates by their association values if you check the **Discount Surrogates** check box and then select the **By Association** radio button. Alternatively, you can discount the improvement measure attributed to each variable in its role as a surrogate by clicking on the

Geometric radio button and entering a value between 0 and 1. CART will use this value to geometrically decrease the weight of the contribution of surrogates in proportion to their surrogate ranking (first, second, third, etc.). Finally, you may click on the **Use Only Top** radio button and select the number of surrogates at each split that you want CART to consider in the calculation.



Misclassification

The Misclassification report shows how many cases were incorrectly classified in the overall tree for both learn and test (or cross-validated) samples. The tables, which can be sorted by percent error, cost or class, display:

Class	Class level
N Cases	Total number of cases in the class
N Misclassified	Total number of misclassified cases in the class
Pct. Error	Percent of cases misclassified
Cost	Fraction of cases misclassified multiplied by cost assigned for misclassification

In our example, we can see that the misclassification errors ranged from one to five percent for the learn sample and two to nine percent for the cross-validated samples, with Class 3 most frequently misclassified in both learn and test data.

Learning Sample					Test Sample				
Class	N Cases	N Mis-Classified	Pct Error	Cost	Class	N Cases	N Mis-Classified	Pct Error	Cost
1	95	2	2.11	0.02	1	95	2	2.11	0.02
2	100	4	4.00	0.04	2	100	5	5.00	0.05
3	98	7	7.14	0.07	3	98	15	15.31	0.15

Prediction Success

The final Summary Report displays the Prediction Success table (also known as the confusion matrix) for both learn and test (or cross-validated) samples. The Prediction Success table shows whether CART tends to concentrate its misclassifications in specific classes and, if so, where the misclassifications are occurring.

The learn and test tables display the following:

Actual Class	Class level
Total Cases	Total number of cases in the class
Percent Correct	Percent of cases for the class that were classified correctly
Class 1 (N)	Number of Class 1 cases classified in each class where N is the total number of cases predicted (correctly or incorrectly) as Class 1
Class 2 (N)	Number of Class 2 cases classified in each class where N is the total number of cases predicted as Class 2
Class 3 (N)	Number of Class 3 cases classified in each class where N is the total number of cases predicted as Class 3

The rest of the table represents a prediction success matrix with rows representing true class assignment and columns representing predicted class assignment.

In our example, we can see that five Class 3 cases in the learn sample were misclassified as Class 2, four Class 2 cases were misclassified as Class 3, and only one Class 1 case was misclassified as Class 3. To switch to the test (cross-validated) sample prediction success table, click on **[Test]** and, similarly, to view row or column percentages rather than counts, click **[Row %]** or **[Column %]**.

Actual Class	Total Cases	Percent Correct	Predicted Class		
			1 N=93	2 N=104	3 N=96
1	95	97.89	93	1	1
2	100	96.00	0	96	4
3	98	92.86	0	7	91
Total:		293.00			
Average:		95.58			
Overall % Correct:		95.56			

Learn Test Class: None Count Row % Column %

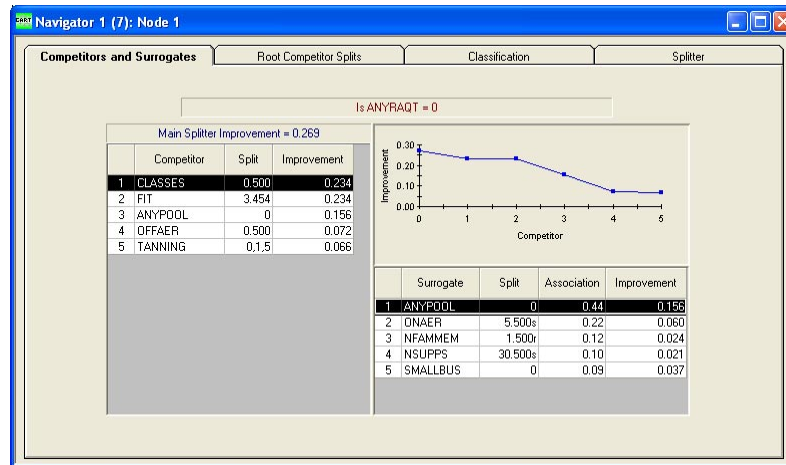
✗ Prediction success tables based on the learn sample are usually too optimistic. You should always use prediction success tables based on the test (or on cross validation, when a separate test sample is not available) as fair estimates of CART performance. CART uses test set performance to find the expected cost and identify the optimal smallest-cost tree.

Detailed Node Reports

To see what else we can learn about our CART trees, return to the Navigator by closing the Summary Reports window or by selecting **Navigator** from the **Window** menu. Move the mouse pointer to the root (top) node in the tree topology panel and click to activate a non-terminal Node Report dialog (or right-click on the root node and select **Node Report**).

The Competitors and Surrogates tab

As illustrated below, the first of the three tabs in the non-terminal node report provides node-specific information for both the competitor and the surrogate splits for the selected node (in this case, the root node).



The splitting rule, *Is ANYRAQT=0*, is displayed in the top line, and the main splitter improvement, the metric CART uses to evaluate the quality of the split, in the following line. A table of the top five competitor splits in decreasing order of importance is displayed in the left panel. Each competitor is identified by a variable name, the value at which the split would be made, and the improvement yielded by the split.

The best competitor, CLASSES, would split at the value 0.500 and would yield an improvement of 0.234, not far below the improvement afforded by the optimal split. The quality of the competitor splits relative to the primary split can also be evaluated by inspecting the line graph displayed in the upper-right panel. The improvement yielded by each competitor split appears on the y-axis and the number or rank of the competitor split on the x-axis, with the primary split improvement displayed at X=0.

The top five surrogates are listed in the bottom-right panel, along with the splitting criterion, the association value, and the improvement yielded by the surrogate split.

In this example, the best surrogate, ANYPOOL, has an association value of 0.439, resulting in an improvement of 0.156 in the misclassification rate.

See the main reference manual for detailed information about how CART calculates and uses Competitors and Surrogates.

See the main reference manual for a detailed discussion of association and improvement.

The Root Competitor Splits tab

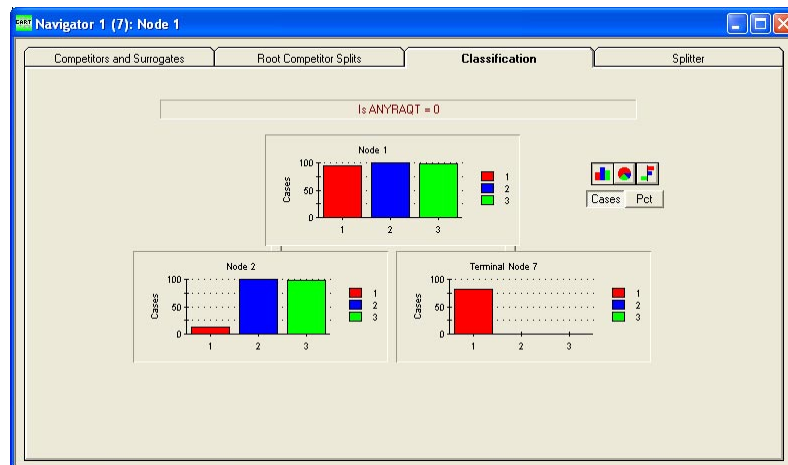
If the root node is selected, the second tab shows the competing root node splits; otherwise this tab is omitted.

Competitor	Split	Improvement	N Left	N Right	N Missing
Main ANYRAQT	0	0.26929	211	82	0
1 CLASSES	0.50000	0.23363	164	129	0
2 FIT	3.45388	0.23363	164	129	0
3 ANYPOOL	0	0.15584	239	54	0
4 OFFAER	0.50000	0.07206	128	165	0
5 TANNING	0.15	0.06622	166	127	0
6 ONAER	5.50000	0.05951	240	53	0
7 SMALLBUS	1	0.03704	15	278	0
8 NFAMMEM	1.50000	0.02424	10	283	0
9 NSUPPS	1.50000	0.02248	25	268	0
10 HOME	1	0.00579	11	282	0

The Classification Tab

The next tab in the non-terminal node report displays node frequency distributions in a bar graph (or, optionally, a pie chart or horizontal bar chart) for the parent-, left -and right-child nodes. If you use a test sample, frequency distributions for learn and test samples can be viewed separately using the **[Learn]** or **[Test]** buttons.

As shown below, the parent node (in this example, the root node) contains all 293 cases. The split, ANYRAQT = 0, is successful in pulling out 82 of the Class 1 observations and putting them in the right-child node, Terminal Node 7. The remaining 13 Class 1 observations and all Class 2 and 3 observations are assigned to the left-child node.



You may switch between counts and percentages by pressing the **[Cases]** or **[Pct]** button.

The horizontal bar chart offers an alternative view of the class partitions. Each colored bar represents one target class. The vertical line shows how the class was partitioned between two children, with the percentage of the class going to the left child shown on the left side and the percentage of the class going to the right child shown on the right side. In this example, less than 20% of Class 1 went to the left side and more than 80% went to the right side.

The Splitter Tab

When a node is split on a categorical variable, an additional tab called "Splitter" is available in the Node Information window for all internal nodes.

For example, declare TANNING as categorical and proceed with the standard GYMTUTOR run introduced above.

The optimal tree now has seven nodes, with node number 5 being split on TANNING. Left-click on this node and choose the Splitter tab.

The screenshot shows the 'Classification' tab for Node 5. At the top, it states 'Is TANNING = 2,3,4,5,6'. Below this is a table with two columns: 'Levels That Go Left' and 'Levels That Go Right'.

Levels That Go Left	Levels That Go Right
2	0
3	1
4	
5	
6	

From this we immediately conclude that all cases with TANNING equal to 2,3,4,5, or 6 go to the left-child node whereas all cases with TANNING equal to 0 or 1 go to the right-child node.

This feature is useful for analyzing high-level categorical splits or when the same categorical variable is used as the main splitter multiple times in a tree.

The Rules tab

The Rules tab is not present in the root node report because there are no rules to display. For our discussion here, we use the Rules tab for Node 5.

The screenshot shows the 'Rules' tab for Node 5. It displays the rule for Node 5 in a text area.

```

/*Rules for node 5*/
if
{
  ANYRAQT == 0
}%%
FIT > 3.45388
{
  node = 5;
  class = 2;
}
  
```

At the bottom, there are two groups of buttons: 'Notation' with 'Classic' and 'SQL' buttons, and 'Probabilities' with 'None', 'Learn', 'Test', and 'Pooled' buttons.

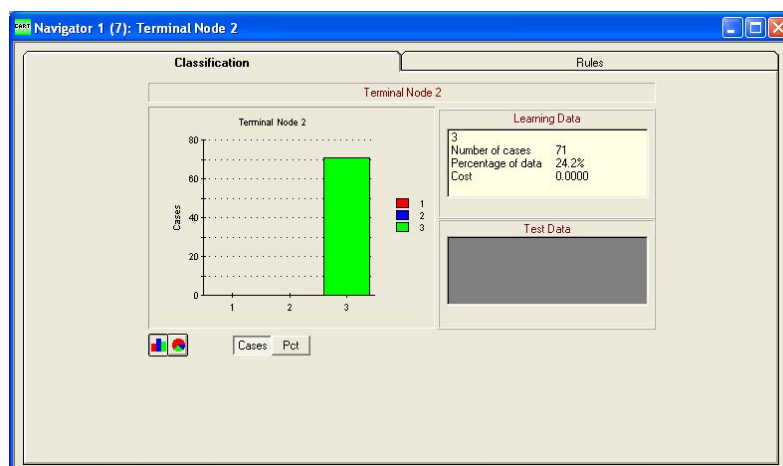
Terminal node reports (with the exception of the root node) contain a Rules dialog that displays the rules for the selected node and/or sub-tree. For example, to view the rules for Node 5, click on the node and select the Rules tab from the Node 5 report dialog. The rules for this node, displayed above, indicate that cases meeting the two specified criteria are classified as Class 2.

To also view learn or test within-node probabilities, click **[Learn]** or **[Test]**. Click **[Pooled]** to view the combined learn and test probabilities.

The rules are formatted as C-compatible code to facilitate applying new data to CART models in other applications. The rule set can be exported as a text file, cut and pasted into another application, and/or sent to the printer. This topic is discussed further below in the section titled "Displaying and Exporting Tree Rules."

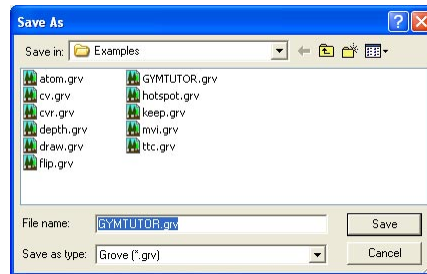
Terminal Node Report

To view node-specific information for a terminal (red) node, click on the terminal node (or right-click and select **Node Report**). A frequency distribution for the classes in the terminal node is displayed as a bar graph (or, optionally, a pie chart), as shown below for the left-most terminal node, Terminal Node 2. Summary node information—class assignment, number of cases in the node, percentage of the data in the node, and misclassification cost—is also displayed for the learn data (and, if you use a test sample, for the test data). In our example, terminal Node 1 is a pure node containing only Class 3 cases and consequently has an associated misclassification cost of zero.



Saving the Grove File

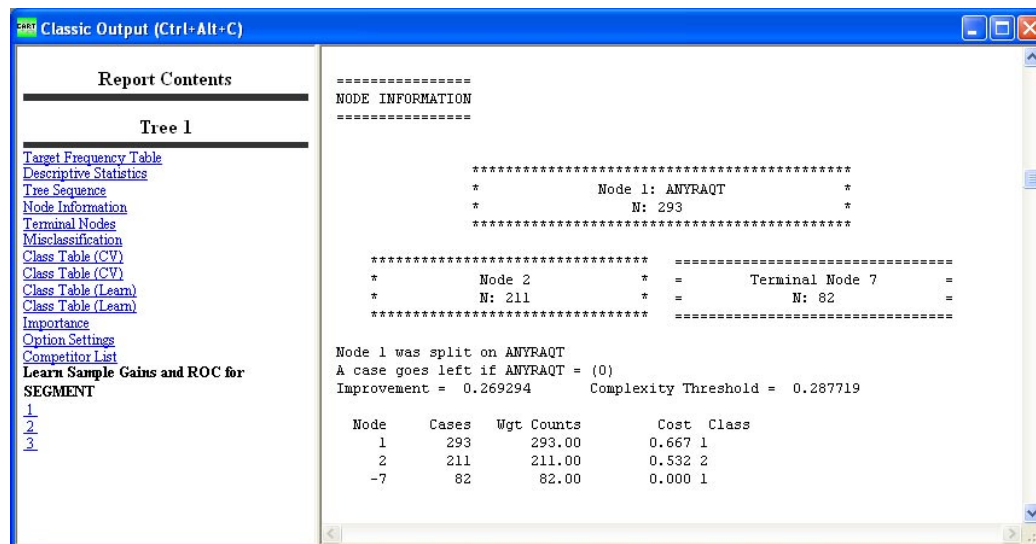
To save the Navigator (aka Grove File) so that you can subsequently reopen the file for further exploration in a later CART session, select **Save Grove...** from the **File->Save...** menu (or press the **[Grove...]** button in the Navigator window). In the **Save As** dialog window, click on the File Name text box to change the default file name (in this case, the data set name, GYMTUTOR). The file extension is by default **.grv**. Specify the directory in which the Grove file should be saved and then click on **[Save]**.



If the trees you are building are large (e.g., over 100 terminal nodes), Windows' system resources can quickly be depleted. To avoid memory problems, be sure to close (or save) any open Navigator windows before generating the next tree. CART will advise you when you are running low on Windows' resources and recommend that you close some of the Navigator windows.

CART Text Output

Now turn to the text output displayed on the CART desktop by closing or minimizing the Node Report and Navigator windows. The outline of the Report Contents for Tree 1, the only tree grown in our example, is displayed in the left panel, as illustrated below. To view a particular section of the output, click on its hyper-link or use the scroll bars to browse the output.



We recommend that you save a copy of the text output as a record of your analysis by selecting **Save CART Output...** from the **File->Save** menu. You can also copy and paste sections of the output into another application or to the clipboard. The font used in the Report window can be changed by selecting **Fonts...** from the **Edit** menu. Use a mono-spaced font such as *Courier* to maintain the alignment of tabular output.

We have already viewed the majority of the text output through the Node Navigator graphical displays. Sections not summarized in the Navigator and Tree Summary Reports include the Variable Statistics and some of the more detailed information in the Tree Sequence and Terminal Node Information tables. For a line-by-line description of these sections, as well as the rest of the text output, consult the main reference manual.

Displaying and Exporting Tree Rules

Non-terminal and terminal node reports (with the exception of the root node) contain a Rules dialog that displays the rules for the selected node and/or sub-tree. For example, to view the rules for Terminal Node 1, click on the node and select the Rules tab from the Terminal Node Report dialog. The rules for this node, displayed below, indicate that cases meeting the four specified criteria are classified as Class 1.


```

/*Terminal node 1*/
/*Rules for root node*/

if
( ( SMALLBUS == 1 ) &&
  ( ANYPOOL == 0 ) &&
  ( ANYRAQT == 0 ) &&
  FIT <= 3.454 )
{
  terminalNode = 1;
  class = 1;}

```

To also view learn or test within-node probabilities, click **[Learn]** or **[Test]**. Click **[Pooled]** to view the combined learn and test probabilities.

The rules are formatted as C-compatible code to facilitate applying new data to CART models in other applications. The rule set can be exported as a text file, cut and pasted into another application, and/or sent to the printer.

To get the set of rules for the entire tree:

1. Select **Rules...** from the **View** menu (or right-click on the root node and select **Rules** from the local menu).
2. Select **Export...** from the **File** menu (a command only available when the Rules dialog is the active window).
3. In the **Save As** dialog specify a directory and file name; the file extension is by default .txt.



This rules display is only intended as a rough guide and does not contain information about surrogate splits. You should use the Translate feature (available by pressing the **[Translate...]** button in the Navigator window) to get the complete representation of the CART model, including surrogates and procedures for handling missing values. See Chapter 7 for details.

Scoring Data

You may score your data by applying any tree reported in the Navigator window. To score your data, proceed as follows:


1. Press **[Score...]** in the Navigator window containing the model you want to apply.
2. In the **Score Data** window:
 - Accept the current data filename or change it using the **[Select...]** button in the **Data** section.
 - Accept the current Grove file (embedded into the current Navigator) or use **[Select...]** to load another one (assuming that it was saved using the **[Save Grove...]** button) in the **Grove** section.

- Check the **Save results to a file** checkbox and specify the output data set name.
 - Choose the tree you want to apply by pressing the **[Select...]** button in the Sub-tree section; by default, CART offers the optimal tree.
 - Set the target, weight, and id variables when applicable.
 - Press **[OK]**.
3. The output data set will contain new variables added by CART, including node assignment, class assignment, and predicted probabilities for each case.



The topics of scoring and translating models are discussed in greater detail in the chapter titled “Scoring and Translating.”


New Analysis

To build another tree using the same data set, select **Construct Model...** from the **Model** menu (or click , the "Model Setup" toolbar icon). CART retains the prior model settings in the Model Setup dialogs.

To use another data set, select **Data File...** from the **File->Open** menu. The new selected file will replace the file currently open and all dialog box settings will return to default values.

✂ If you want to ensure that all default settings are reset to their original state, select **Clear Workspace** from the **File** menu.

Saving Command Log

To save the Command Log, select **Open Command Log...** from the **View** menu (or press , the "Command Log" toolbar icon) and then select **Save** from the **File** menu. Specify a directory and the name of the command file, saved by default with a .CMD extension.

If your model is rather time-consuming (e.g., the model contains many candidate predictors, most of which are categorical), saving the command log can expedite further manipulation of model setup specifications in subsequent CART sessions.

See Chapter 13 for more about the CART command log and running CART in batch mode.

✂ **IMPORTANT NOTE!** When a CART session is finished (the CART application is closed), a log file containing all commands issued during the session is created in the CART temporary folder (specified in **Edit->Options->Directories**). This text file is given a name that starts with "CART" followed by month and day, followed by hour (military convention 0:23), minutes, and seconds, followed by two underscores. For example, CART1101173521__.TXT refers to the CART session that was finished on November 1st, at 5:35:21 pm. This serves as a complete audit trail of your work with the CART application. Also note that renaming a log file to *.CMD while subsequently submitting (**File->Submit Command File...**) in a new CART session will essentially reproduce the entire previous CART session.

💡 There is no limit to the number of session command logs that are saved to the CART temporary files folder. We suggest that you regularly clean up this folder by deleting obsolete files.

Chapter

12



Features and Options

*This chapter provides information on additional
features and options found in CART*

Features and Options

This chapter provides an orientation to the features and options not covered in the previous chapters, as well as a description of CART's more advanced options. If any terms or concepts are new to you, please consult the main reference manual.

Unsupervised Learning and Cluster Analysis

CART in its classification role is an excellent example of "supervised" learning: you cannot start a CART classification analysis without first selecting a target or dependent variable. All partitioning of the data into homogenous segments is guided by the primary objective of separating the target classes. If the terminal nodes are sufficiently pure in a single target class the analysis will be considered successful even if two or more terminal nodes are similar on most predictor variables.

Unsupervised learning, by contrast, does not begin with a target variable. Instead the objective is to find groups of similar records in the data. One can think of unsupervised learning as a form of data compression: we search for a moderate number of representative records to summarize or stand in for the original database.

Consider a mobile telecommunications company with 20 million customers. The company database will likely contain various categories of information including customer characteristics such as age and postal code, product information describing the customer's mobile handset, features of the plan the subscriber has selected, details of the subscriber's use of plan features, and billing and payment information. Although it is almost certain that no two subscribers will be identical on every detail in their customer records, we would expect to find groups of customers who are similar in their overall pattern of demographics, selected equipment, plan use, and spending and payment behavior. If we could find, say, 30 representative customer types such that the bulk of customers are well described as belonging to their "type," this information could be very useful for marketing, planning, and new product development.

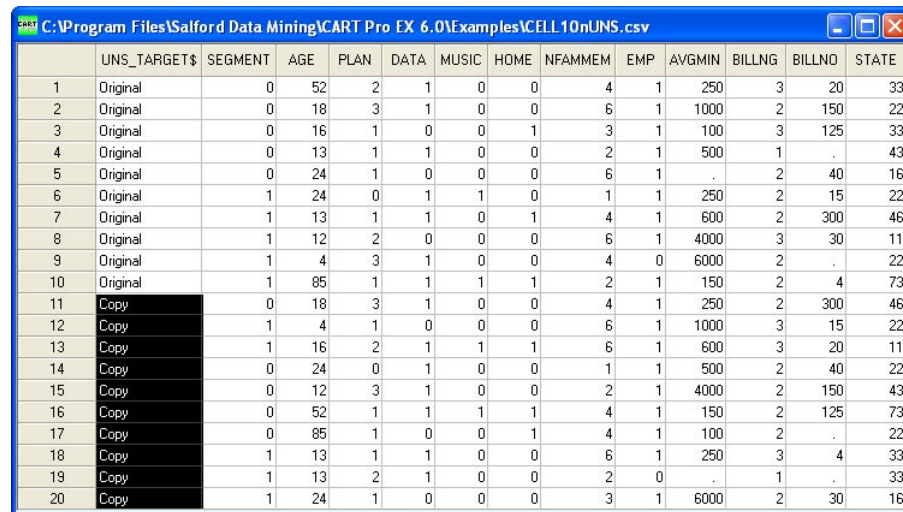
We cannot promise that we can find clusters or groupings in data that you will find useful, but we include a method quite distinct from that found in other statistical or data mining software. CART and other Salford data mining modules now include an approach to cluster analysis, density estimation and unsupervised learning using ideas that we trace to Leo Breiman, but which may have been known informally among statisticians at Stanford and elsewhere for some time. The method detects structure in data by contrasting original data with randomized variants of that data. Analysts use this method implicitly when viewing data graphically to identify clusters or other structure in data. Take, for example, customer ages and handsets owned. If there were a pattern in the data, we would expect to see certain handsets owned by people in their early 20s and rather different handsets owned by customers in their

early 30s. If every handset is just as likely to be owned in every age group then no structure relates these two data dimensions. The method we use generalizes this everyday detection idea to higher dimensions.

The method consists of the following steps:

1. Make a copy of the original data, and then reorder the data in each column using a random scramble. Do this one column at a time, using a different random ordering for each column, so that no two columns are scrambled in the same way.

As an example, starting with data typical of a mobile phone company, suppose we randomly exchange date of birth information in our copy of the database. Thus, each customer record would likely contain age information belonging to another customer. We now repeat this process in every column of the data. Breiman uses a variant in which each column of original data is replaced with a bootstrap resample of the column; either method can be used in Salford's software. The following displays a small example.



	UNS_TARGET\$	SEGMENT	AGE	PLAN	DATA	MUSIC	HOME	NFAMMEM	EMP	AVGMIN	BILLNG	BILLNO	STATE
1	Original	0	52	2	1	0	0	4	1	250	3	20	33
2	Original	0	18	3	1	0	0	6	1	1000	2	150	22
3	Original	0	16	1	0	0	1	3	1	100	3	125	33
4	Original	0	13	1	1	0	0	2	1	500	1	.	43
5	Original	0	24	1	0	0	0	6	1	.	2	40	16
6	Original	1	24	0	1	1	0	1	1	250	2	15	22
7	Original	1	13	1	1	0	1	4	1	600	2	300	46
8	Original	1	12	2	0	0	0	6	1	4000	3	30	11
9	Original	1	4	3	1	0	0	4	0	6000	2	.	22
10	Original	1	85	1	1	1	1	2	1	150	2	4	73
11	Copy	0	18	3	1	0	0	4	1	250	2	300	46
12	Copy	1	4	1	0	0	0	6	1	1000	3	15	22
13	Copy	1	16	2	1	1	1	6	1	600	3	20	11
14	Copy	0	24	0	1	0	0	1	1	500	2	40	22
15	Copy	0	12	3	1	0	0	2	1	4000	2	150	43
16	Copy	0	52	1	1	1	1	4	1	150	2	125	73
17	Copy	0	85	1	0	0	1	4	1	100	2	.	22
18	Copy	1	13	1	1	0	0	6	1	250	3	4	33
19	Copy	1	13	2	1	0	0	2	0	.	1	.	33
20	Copy	1	24	1	0	0	0	3	1	6000	2	30	16

Note that all we have done is to move information about in the "Copy" portion of the database. Other than moving data we have not changed anything (discrete levels or values), so aggregates such as averages and totals will not have changed. Any one-customer record is now a "Frankenstein" record, with all items of information having been obtained from a different customer. In the above example, "Copy #17" has been given AGE=85 from customer #10, and the average bill (AVGBILL) from customer #3.

2. Now append the scrambled data set to the original data. We therefore now have the same number of columns as before but twice as many rows. The top portion of the data is the "Original" data and the bottom portion will be the scrambled "Copy."

Add a new column to the data to label records by their data source ("Original" vs. "Copy").

3. Generate a predictive model to attempt to discriminate between the Original and Copy data sets. If it is impossible to tell, after the fact, which records are original and which are random artifacts then there is no structure in the data. If it is easy to tell the difference then there is strong structure in the data.

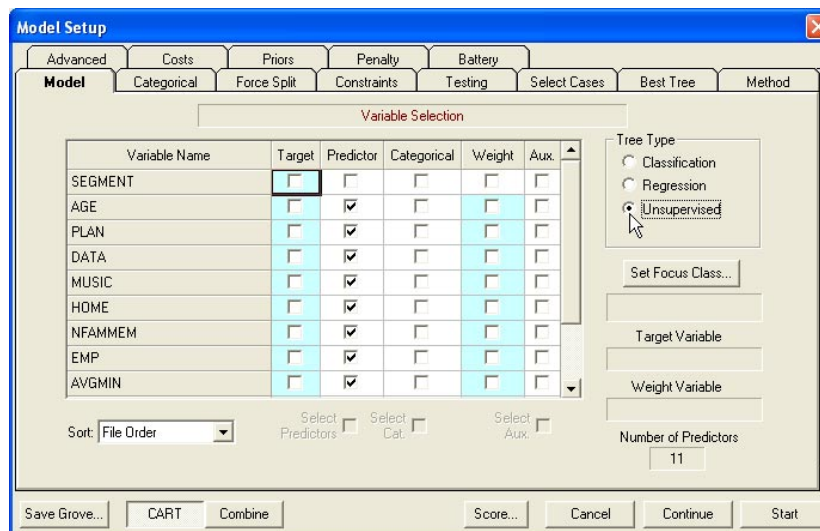
4. In the CART model separating the Original from the Copy records, nodes with a high fraction of Original records define regions of high density and qualify as potential "clusters." Such nodes reveal patterns of data values that appear frequently in the real data but not in the randomized artifact.

We do not expect the optimal-sized tree for cluster detection to be the most accurate separator of Original from Copy records. We recommend that you prune back to a tree size that reveals interesting data groupings.

Setting Up an Unsupervised Model

To set up an unsupervised model we use the **Model Setup—Model** tab. Start by defining your predictors using the check boxes in the **Predictors** column. For unsupervised learning there is no target variable. If a target variable is checked it will be discarded and ignored.

The only other setup required is to select the **Unsupervised** radio button from the control section titled **Tree Type**. As you can see, all the other Model Setup tabs remain available for additional controls that the analyst may desire.



- ✎ If we simply scramble the data without resampling then the summary statistics for the Original and Copy data sets must be identical. The scrambling destroys any correlation structure in the data (linear or nonlinear). Hence, when using all the data for training no variable can split the data productively in the root node (which is as it should be). If the data sets can be separated at all, a combination of at least two variables will be required. Thus, in the telecommunications example, the average customer age is of course identical in the original and the copy. But the average age of customers having iPhones may very well not be equal across Original and Copy datasets.

If it is not possible to develop a good model to separate Original and Copy data, this means that there is little structure in the Original data and there are no distinctive patterns of interest.


This approach to unsupervised learning represents an important advance in clustering technology because

- a) variable selection is not necessary and different clusters may be defined on different groups of variables.
- b) preprocessing or rescaling of the data is unnecessary as these clustering methods are not influenced by how the data are scaled.
- c) the missing values present no challenges because the methods automatically manage missing data.
- d) the CART-based clustering gives easy control over the number of clusters and helps select the optimal number.

The Force Split tab

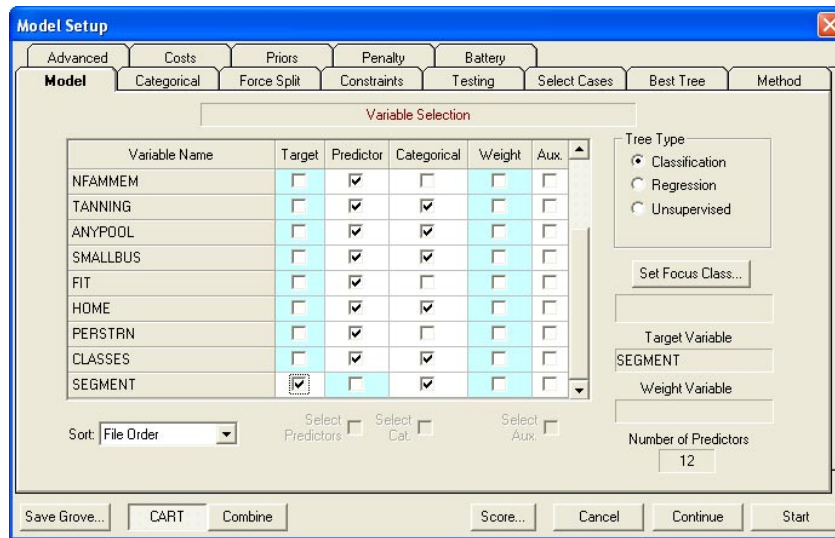
The **Model Setup—Force Split** tab is new in CART 6.0. This setup tab allows you to dictate the splitter to be used in the root node (primary splitter), or in either of the two child nodes of the root. Users wanting to impose some modest structure on a tree frequently desire this control. More specific controls also allow the user to specify the split values for both continuous and categorical variables if you prefer to do so.

Specifying the Root Node Splitter

For this example we once again will be using the GYMTUTOR.CSV data file. To specify the target variable, use the  to scroll down the variable list until SEGMENT is visible. Put a checkmark inside the checkbox located in the **Target** column.

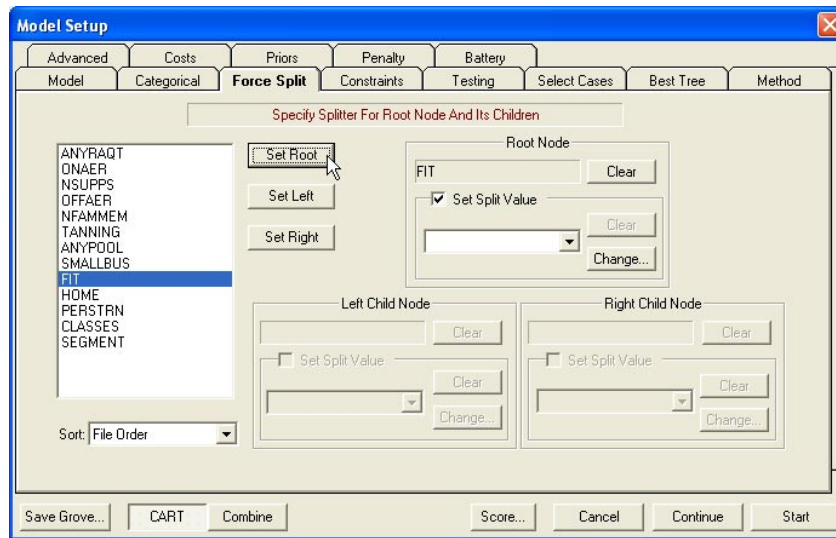
Select the remaining variables and place a checkmark in the **Predictors** column. Also, place checkmarks in the **Categorical** column against those predictors that should be treated as categorical. For our example, specify ANYRAQT, TANNING, ANYPOOL, SMALLBUS, HOME, and CLASSES as categorical predictor variables.

The resulting **Model Setup** tab will look like the following.



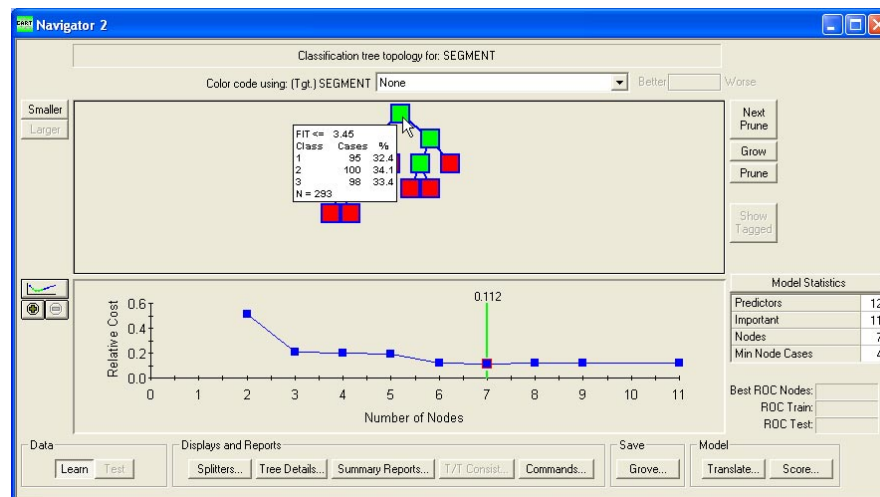
Now let's take a look at the **Model Setup—Force Split** tab and specify a root node split. In this example we only want to force a split on a specific variable without concern for the split value itself. Later we will force a split variable and value.

To specify the root node split, select FIT from the variable list and click the **[Set Root]** button. This tells CART that the root node split must use the variable FIT for the initial split even if it is not the optimal splitter. The resulting dialog appears as follows.



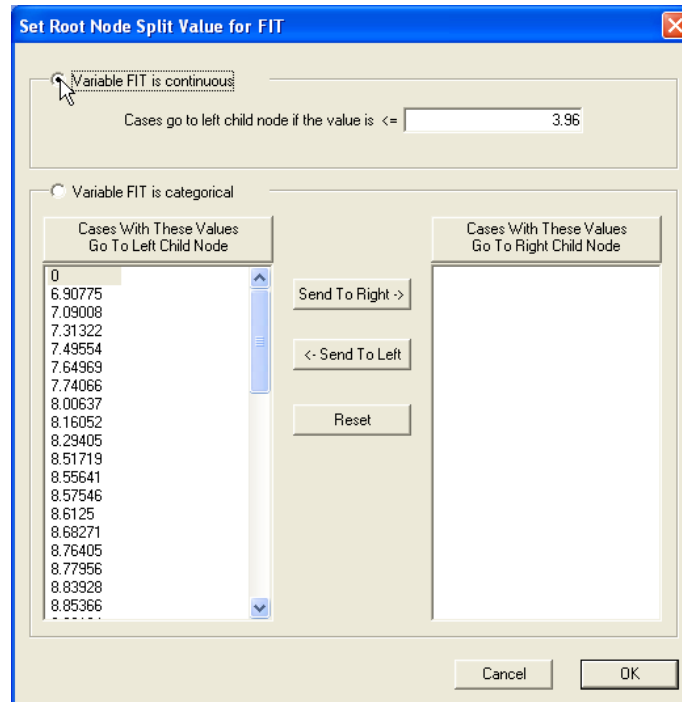
Keeping all other default settings, click **[Start]** to build the model.

As you can see, by hovering the mouse over the root node, the resulting Navigator indeed splits on the variable FIT in the root node with a split point of 3.45388.



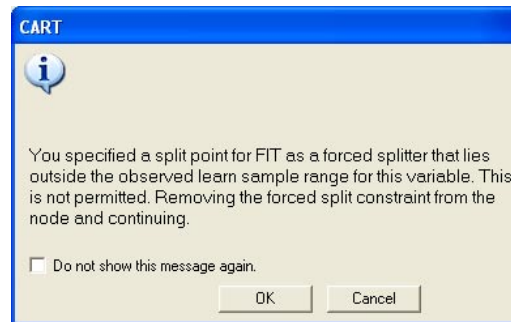
Now let us show a similar example, except here we specify the split point as well. In our previous example we saw the root node split of $FIT \leq 3.45388$. In this example we will force the split on FIT's mean value of 3.96.

To do so, return to the **Model Setup—Force Split** tab. The previous specified variable FIT should be retained and displayed as the Root Node entry. This time we will check **[x] Set Split Value** and then click the **[Change...]** button. The resulting **Set Root Node Splits Value** dialog will appear.

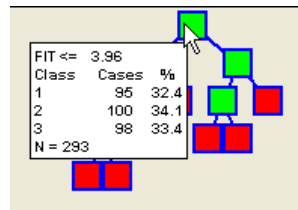


This dialog allows you to specify the split value for continuous variables in the upper portion, and categorical variables in the lower portion. Here we have placed the value 3.96 in the entry box titled "Cases go to left child node if the value is <=". Click **[OK]** to continue and return to the Model Setup dialog. From the Model Setup window, click **[Start]** to build the model.

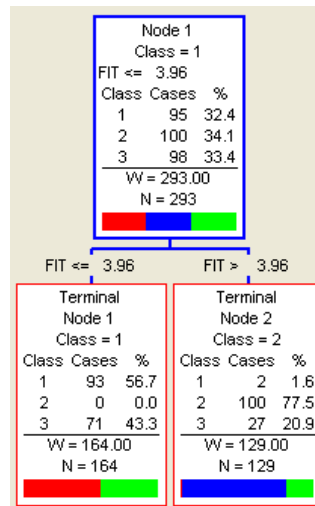
- The user is allowed to enter any value as long as it falls within the range of permissible values. In the case of the variable FIT, the minimum value is zero and the maximum is 10.127. However, the user who enters a value outside the range would receive an error like the following:



From the resulting navigator, if you hover your mouse over the root node, we can see that CART now uses both the specified variable FIT and the split point 3.96.



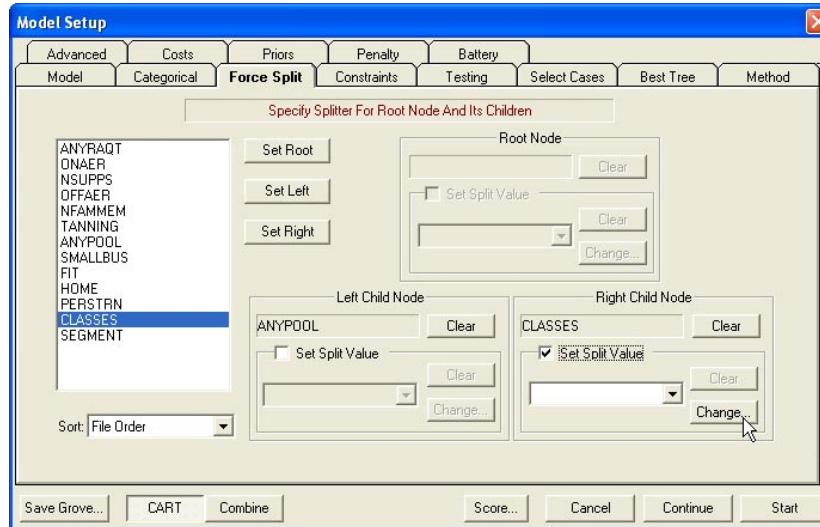
An alternative view would be to look at the tree details diagram by clicking the **[Tree Details...]** button found on the Navigator. This would give you the following view, again showing that the split variable and the value were utilized.



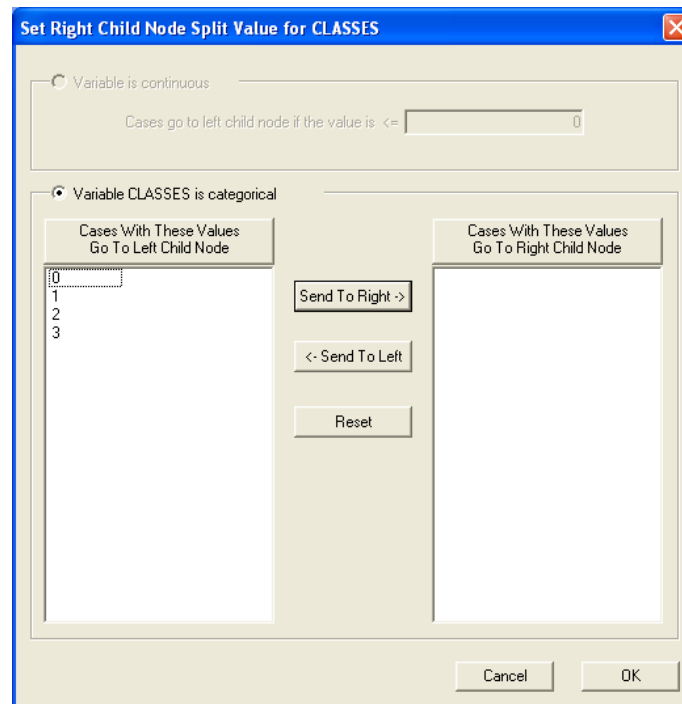
Specifying the Left/Right Child Node Splitter

Using the same root node force split variable and value we now demonstrate how to specify the right/left child node splits. Like the root node split, the user can specify not only the variable, but also a split value.

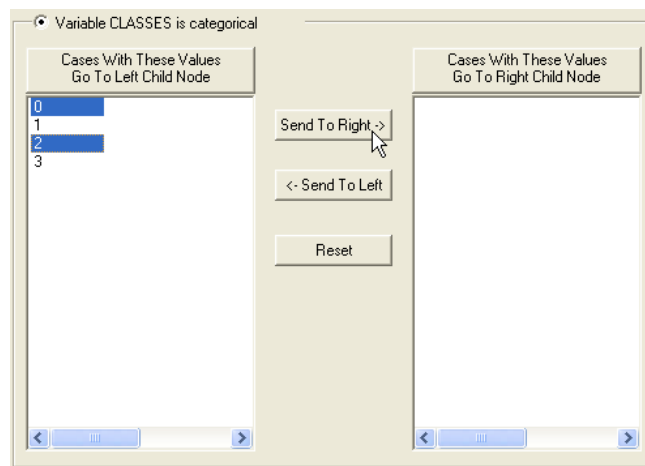
In this example we use the categorical variables ANYPOOL (0, 1) and CLASSES (0, 1, 2, 3). Using the **[Set Left]** button, select ANYPOOL as the Left Child Node splitter. Repeat using the **[Set Right]** button for CLASSES. Because ANYPOOL is a binary, no split value is specified. For the Right Child Node, check **[x] Set Split Value** and then click the **[Change...]** button.



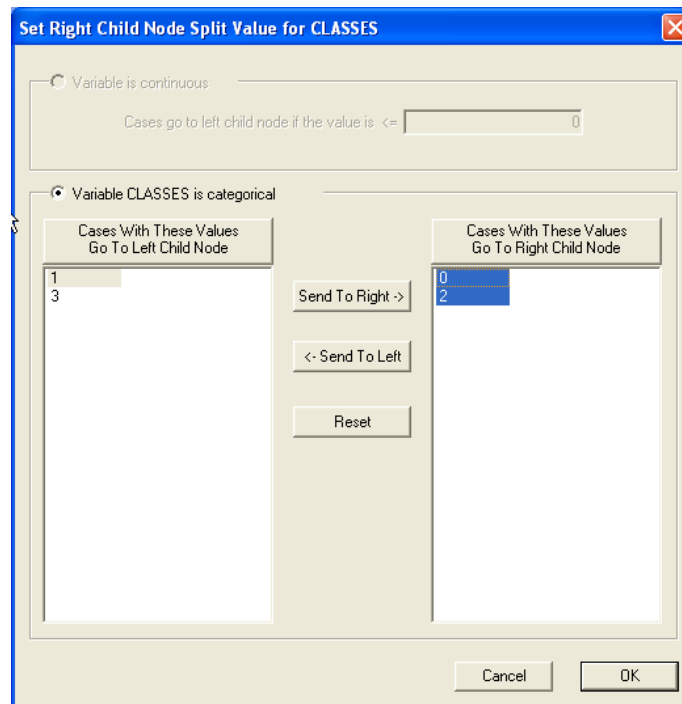
The resulting **Set Root Node Splits Value** dialog will appear.



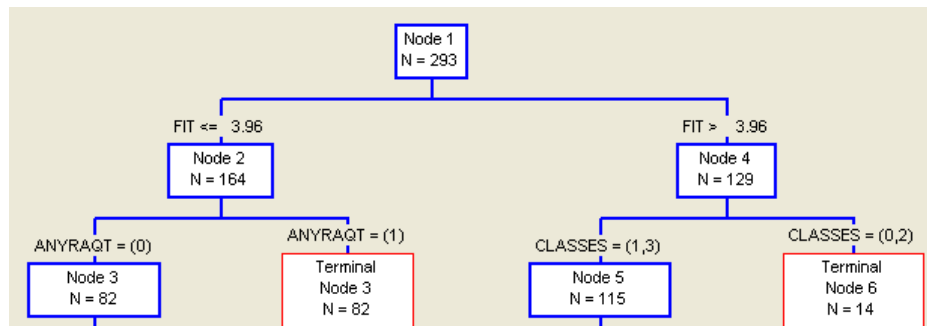
Unlike our previous example for continuous variables, this time we are using the lower portion of the dialog to specify the left/right direction for individual classes. To do so, select the classes you want to go left or right and then click either the **[Send To Right->]** or the **[<-Send To Left]** button.



In this example we are choosing to send classes 1 and 3 to the left, and classes 0 and 2 to the right. The resulting setup dialog looks as follows.



Click **[OK]** to continue and return to the Model Setup dialog. From the Model Setup window, click **[Start]** to build the model. From the resulting Navigator, if we click on the **[Tree Details...]** button, we will see that our specified forced splits have been implemented. For illustrative purposes we are only displaying the top two level splits.





Command-line users will use the following command syntax to set the force split rules:

```
FORCE ROOT|LEFT|RIGHT ON <predictor> AT <splits>
```

For example:

```
FORCE ROOT ON GENDER$ AT "Male", "Unknown"
FORCE LEFT ON REGION AT 0,3,4,7,999
FORCE RIGHT ON INCOME AT 100000
```

To reset forced splits, use the command with no options

```
FORCE
```



The Constraints tab

The **Model Setup–Constraints** tab is new in CART 6.0. This setup tab specifies how predictor variables are constrained for use, as primary splitters and/or as surrogates, at various depths of the tree and according to the size of the learn sample in the node.

By default, all predictors are allowed to be used as primary splitters (i.e., competitors) and as surrogates at all depths and node sizes. The Constraints tab is used to specify at which depths and in which partitions (by size) the predictor, or group of predictors, is not permitted to be used, either as a splitter, a surrogate, or both.

Constraints and Structured Trees

In marketing applications we often think about predictors in terms of their role in influencing a consumer's choice process. For example, we distinguish between characteristics of the consumer, over which the marketer has no control, and characteristics of the product being offered, over which the marketer may have some degree of control.

Normally CART will be unaware of the different strategic roles different variables may play within the business context and a CART tree designed to predict response will mix variables of different roles as needed to generate an accurate predictive model. However, it will often be useful to be able to **STRUCTURE** a CART tree so that there is a systematic order in which the variables enter the tree.

For example, we may want the tree to use only characteristics of the consumer at the top of the tree and to have only the bottom splits based on product characteristics.

Such trees are very easy to read for their strategy advice: first they segment a database into different types of consumer, and then they reveal the product configurations or offers that best elicit response from each consumer segment.

CART now offers a powerful mechanism for generating structured trees by allowing you to specify where a variable or group of variables are allowed to appear in the tree. The easiest way to structure a tree is to group your predictor variables into lists and then to dictate the levels of the tree where each list is permitted to operate. Thus, in our marketing example, we could specify that the consumer attributes list can operate anywhere in the top four levels of the tree (but nowhere else) and that the product attributes list can operate from level five and further down into the tree (but nowhere else). Structuring a tree in this way will provide the marketer with exactly the type of tree described above.

How did we know to limit the consumer attributes to the first four levels? We know only by experimenting by running analysis using different ways to structure the tree. If we are working with two groups of variables and want to divide the tree into top and bottom regions, we can try dividing the tree at different depths, for example, by enforcing the top/bottom division point at a depth of 2, then 3, then 4, etc. Usually, it is quickly apparent that one of these divisions works better than the others.

How should the variables be divided into different lists? This is entirely up to the analyst, but typically each list will represent a natural grouping of variables. You might group variables by the degree of control you have over them, by the cost of acquisition, by accepted beliefs regarding their importance, or for convenience.

Example: In a model of consumer choice we wanted to develop a model relating consumer needs and wants to a specific product being selected. An unrestricted CART model always placed the country of origin of the product in the root node as our consumers for the product in question had very strong feelings on this subject. For a number of reasons our client wanted the country of origin to be the LAST splitter in the tree. To generate such a tree was easy using CONSTRAINTS: we created one list of attributes containing all the consumer wants and needs and specified that those variables could only be used in the top region of the tree. We also created another list consisting of just the one country of origin attribute and specified that it could only appear in the bottom portion of tree. The resulting tree was exactly what the marketers were looking for.

We use marketing as an example because it is easy for most readers to understand, but constraints for structuring trees can be used in many applications. In scientific applications, constraints may be imposed to reflect the natural or causal order in which certain factors may be triggered in a real world process. Constraints may also be used to induce a tree to use broad general predictors at the top and then to complete the analysis using more specific and detailed descriptors at the bottom.

CART allows you to structure your trees in a number of ways. You can specify where a variable can appear in the tree based on its location in the tree or based on the size of the sample arriving at a node. You can also specify as many different regions in the tree as you wish. For example, you could specify a different list for every level of the tree, and one predictor may appear on many different lists.

Structured Trees Using Predictor Groups

For the following example we once again will use the GYMTUTOR.CSV data file used in the Chapter 11 segmentation example. Using the **Model Setup—Model** tab, specify the target variable as SEGMENT by placing a checkmark inside the checkbox located in the **Target** column.

Select the remaining variables and place a checkmark in the **Predictors** column. Also place checkmarks in the **Categorical** column against those predictors that should be treated as categorical. For our example, specify ANYRAQT, TANNING, ANYPOOL, SMALLBUS, HOME, and CLASSES as categorical predictor variables.

The resulting **Model** tab will look like the following.

Model Setup

Advanced Costs Priors Penalty Battery

Model Categorical Force Split Constraints Testing Select Cases Best Tree Method

Variable Selection

Variable Name	Target	Predictor	Categorical	Weight	Aux.
NFAMMEM	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TANNING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ANYPOOL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SMALLBUS	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
FIT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
HOME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PERSTRN	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CLASSES	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SEGMENT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Sort: File Order

Select Predictors ☐ Select Cat. ☐ Select Aux. ☐

Tree Type

- ☒ Classification
- ☐ Regression
- ☐ Unsupervised

Set Focus Class...

Target Variable

SEGMENT

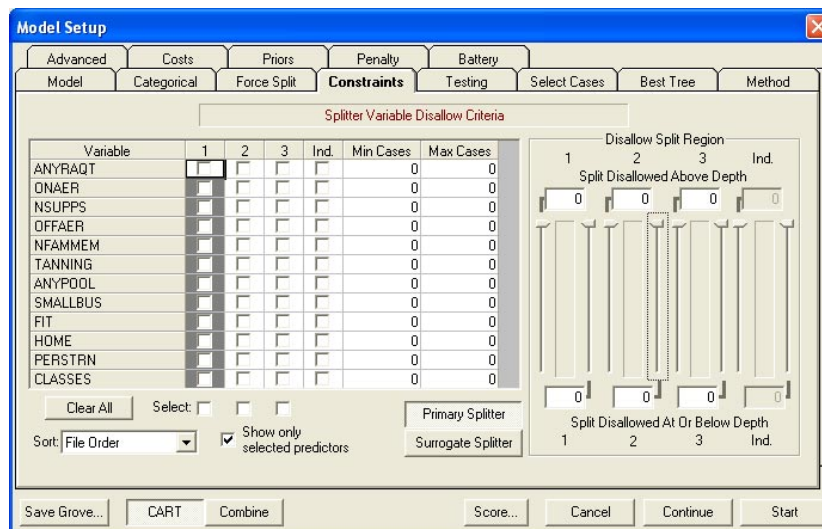
Weight Variable

Number of Predictors

12

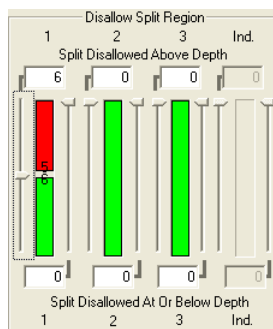
Save Grove... CART Combine Score... Cancel Continue Start

Let's take a closer look at the **Model Setup—Constraints** tab and get ready to specify a group of constraints.

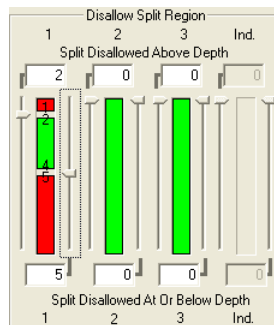


The **Constraints** tab has two main sections. In the left pane we can specify groups of variables using the check boxes in the columns labeled “1,” “2,” or “3.” The column labeled “Ind.” is used for ungrouped, or individual, variables.

The second main section titled **Disallow Split Region** has a set of sliders used to specify constraints for each of the three groups, or individual variables. The sliders come in pairs, (one on the left and one on the right). The left slider controls the “Above Depth” value, while the right slider controls the “Below Depth” value. As the sliders are positioned, either a green or red color-coding will appear indicating at what depth a variable is allowed or disallowed as a splitter. In the following screen, a group-1 constraint has set on the “Above Depth.” Here the slider and color-coding indicates the group-1 variables are disallowed (red) above the depth of 6, but permitted (green) at any depth greater than or equal to 6.



A more complex example would be setting both the above and below constraints on a group of variables. In the next screen we use the left slider to specify our “Above Depth” constraint of 2, and the right slider to specify our “Below Depth” constraint of 5. Now our selected variable(s) are only permitted for the depth levels of 2, 3, or 4. They are disallowed above 2 and below 5.



Now let's run an example and specify two groups of structure constraints using the GYMTUTOR.CSV data. One group of variables is the consumer characteristics, and a second group is the or product characteristics.

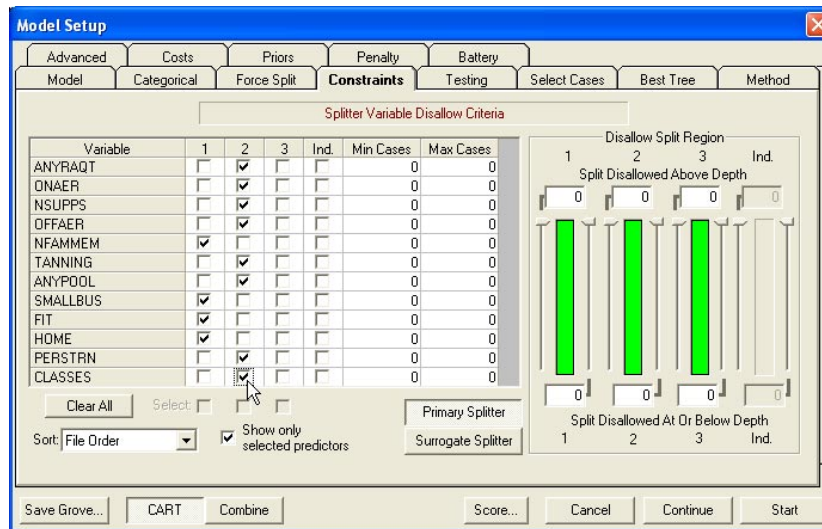
Consumer characteristics:

- NFAMMEM** Number of family members
- SMALLBUS** Small business discount (binary indicator coded 0, 1)
- FIT** Fitness score
- HOME** Home ownership (binary indicator coded 0, 1)

Product characteristics:

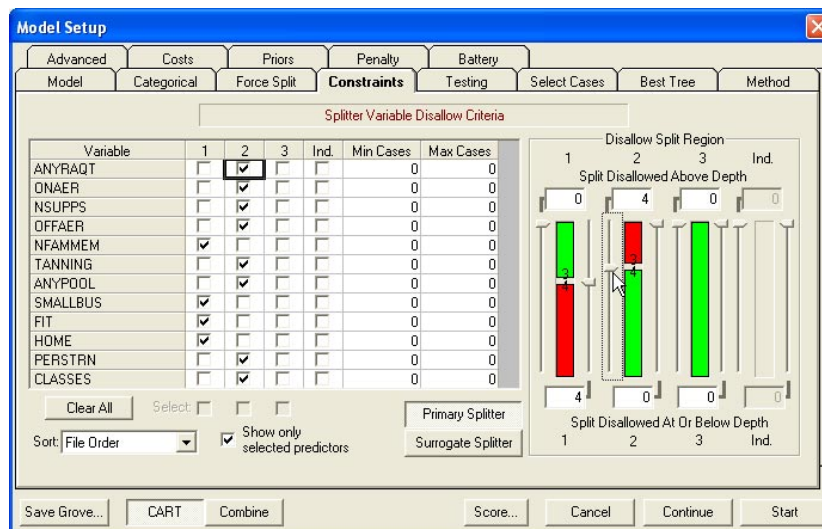
- ANYRAQT** Racquet ball usage (binary indicator coded 0, 1)
- ONAER** Number of on-peak aerobics classes attended
- NSUPPS** Number of supplements purchased
- OFFAER** Number of off-peak aerobics classes attended
- TANNING** Number of visits to tanning salon
- ANYPOOL** Pool usage (binary indicator coded 0, 1)
- PERSTRN** Personal trainer (binary indicator coded 0, 1)
- CLASSES** Number of classes taken

For our group-1 variables, place a check mark for each using the column labeled “1.” Repeat this process for group-2 using the column labeled “2.” The resulting **Constraints** tab will look as follows.

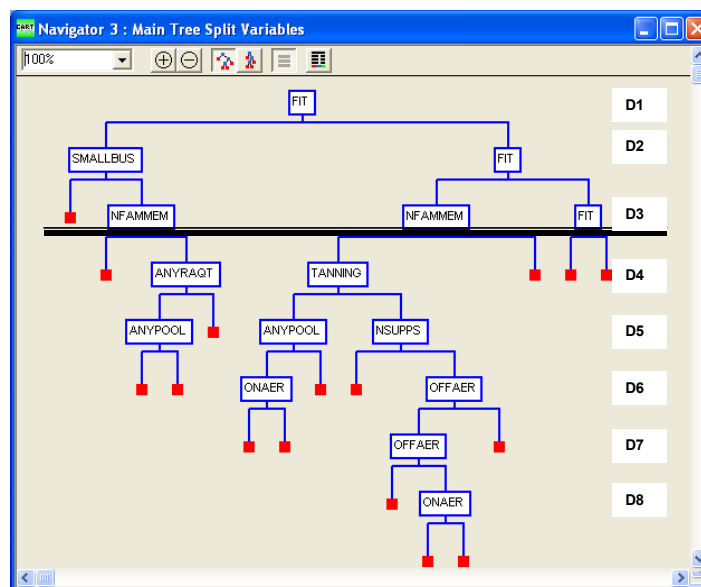


Next we use the slider controls in the **Disallow Split Region** to specify the depth (above and below) where our two groups will be allowed in the tree.

For group-1, we use the right slider control to disallow splits below the depth of 4. For group-2, we use the left slider to disallow splits above the depth of 4. In other words, the group-1 consumer variables should only be split in the top portion of the tree, while the group-2 product variables should only be found in the lower portions of the tree. The resulting setup looks as follows.



Let's run an exploratory tree with the above constraints and view the splitters. As you can see below, the defined constraints for both groups were implemented. None of the group-1 variables are below the depth of three (D3), and none of the group-2 variables are found above the depth of four (D4).



Learn Sample Size

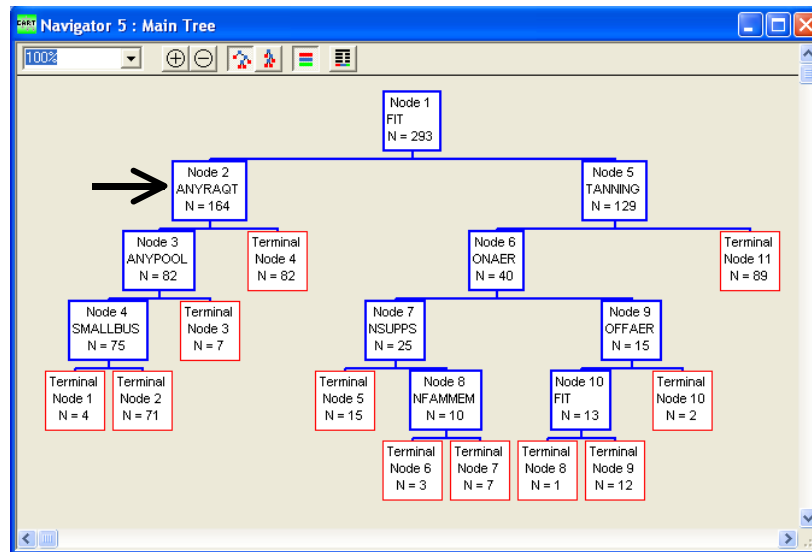
CART also allows the user to constrain a tree according to the size of the learn sample in the nodes. Instead of using depth to control where a splitter can be used, we disallow splits based on the size of the learn sample in the node. The "Min Cases" and "Max Cases" columns are used to enter positive values in the cells.

- ♦ Min Cases - variable will not be used if the node has more than the specified number of records.
- ♦ Max Case - variable will not be used if the node has fewer than the specified number of records.

In the following example we constrain ANYRAQT from being used as a splitter unless there are fewer than 200 learn sample observations in a node.

Variable	1	2	3	Ind.	Min Cases	Max Cases
ANYRAQT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	200	0
ONAER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
NSUPPS	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
OFFAER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
NFAMMEM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
TANNING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
ANYPOOL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
SMALLBUS	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
FIT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
HOME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
PERSTRN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0
CLASSES	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	0

Had we left the tree unconstrained, ANYRAQT would have been the first split in the tree. However, as we can see from the tree details, the constraint was implemented and ANYRAQT does not appear as a splitter until Node 2 with only 164 observations.



Command-line users will use the following command syntax to set the constraints:

```
DISALLOW <variable list> [ / ABOVE=<depth>, BELOW=<depth>,
                           MORE=<node_size>, FEWER=<node_size>, SPLIT ]
```

For example:

```
DISALLOW OFFAER / ABOVE = 3 SPLIT
DISALLOW NFAMMEM / BELOW = 4 SPLIT
```

```
DISALLOW ANYRAQT / MORE = 200 SPLIT
DISALLOW CLASSES / FEWER = 25 SPLIT
```


To reset constraints, use the command with no options

DISALLOW

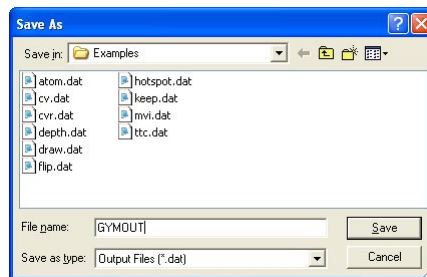
Saving and Printing Text Output

By default, CART text output is sent to the Report window. If you would like to save or print results, use one of the following methods.

Specify Output File Prior to Processing

To simultaneously save the text output to a file, you must specify the output file prior to processing. Once the output file is specified, all subsequent output will be recorded in the selected file.

1. Select **Log Results to...** from the **File** menu and choose the **File...** option.
2. Click on the **File Name** text box in the Text Results to File dialog box to set the file name, as illustrated below.
3. Select the directory in which the file should be saved.
4. Click on **[Save]**.



To stop sending the output to a file, select the **Log Results to: Window** from the **File** menu.

- ✗ The **CART Output** window must be active to have access to the above menus.
- ✗ Due to some features of the operating system, you will not be able to see the contents of the log file until after CART is closed, a new log file is specified, or the output is logged back to Window.



Command-line equivalents

```
OUTPUT "<file_name.dat>"  
OUTPUT *
```

Specify Output File Post-Processing

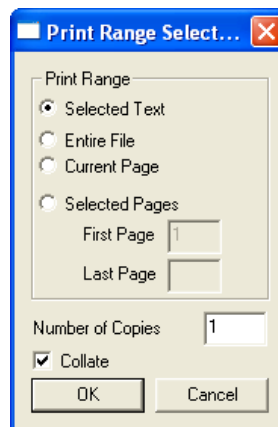
To save the complete current contents of the **CART Output** window to a file after you have built a tree:

1. Select **Save Output...** from the **File->Save**.
2. Click on the **File Name** text box in the Text Results to File dialog box to set the file name, as illustrated below.
3. Select the directory in which the file should be saved.
4. Click on **[Save]**.

To save a particular section of the output, highlight that section and select **Copy** from the **Edit** menu (or from the toolbar). Paste the copied text to the Notepad by selecting **New Notepad...** from the **File** menu and then save the notepad contents by selecting **Save As...** from the **File** menu. Alternatively, after you copy the text, paste it to another application such as Microsoft Word or Excel.

Printing the CART Output Window

To send output contained within the CART Output window, simply select **Print...** from the **File** menu. The following Print dialog will appear and provide a set of Print Range options. Choose the desired option and click the **[OK]** button to complete printing.



Memory Management

Formerly, CART was compiled into distinct memory versions (64MB, 128MB, etc). A user's license determined which memory version was delivered. Thus, the license was tied to the amount of workspace inherent in the program and (loosely) tied to the amount of data, type of data (categorical vs. continuous), size of final tree, etc., that the user could analyze.

Licensing and workspace are handled differently in CART 6. A user's license sets a limit on the amount of learn sample data that can be analyzed. The learn sample consists of the data used to grow the maximal tree. Note that the number of test sample data points that may be analyzed is unlimited.

For example, suppose you are using our 32MB version that sets a learn sample limitation of 8 MB. Each data point occupies 4 bytes. An 8MB license will allow up to $8 * 1024 * 1024 / 4 = 2,097,152$ learn sample data points to be analyzed. A data point is represented by 1-variable by 1-observation (1-row by 1-column).

💡 In general, the analysis workspace provided to build the tree will be adequate for "most" modeling scenarios. However, if the user models a large number of high-level categorical predictors, or is using a high-level categorical target, the user may encounter workspace limitations that will not allow the entire learn sample to be used. In these special cases the user will have to upgrade to a larger memory version, or use one of the options discussed below.

Workspace Usage

Because CART checks on every possible split at every node, CART must store the full data set in memory when it is building a tree. In certain situations it may be necessary to restrict the size of the maximal tree grown so the analysis will fit into the workspace available on your computer.

If the available workspace is not large enough to grow the requested tree, a CURRENT MEMORY REQUIREMENTS table will appear in the CART Report window that looks something like the following:

```
CURRENT MEMORY REQUIREMENTS
TOTAL: 41492578. DATA: 2223939. ANALYSIS: 41492578.
AVAILABLE: 33750000. DEFICIT: 7742578.
```

```
=====
CART has insufficient memory to continue.
Try sub-sampling nodes with the command: LIMIT SUBSAMPLE
=====
```

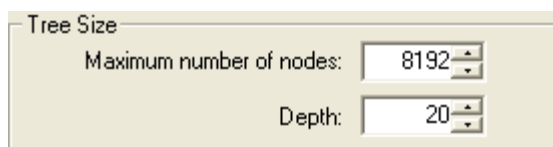
If this occurs, or if you suspect the problem is too large for the workspace, you may need to specify limitations on the structure of the tree to be able to process the model.

Memory Usage Example

A data set with 32,231 records, a 10-level target categorical variable, and 68 categorical predictors is used to illustrate how to overcome a memory shortfall. As shown below, the top three rows provide an overview of the workspace requirements for this example. The estimated total workspace is 41,357,617 elements, 2,092,770 elements to hold the data and 39,264,847 to process the analysis. Because the available workspace is only 33,750,000 workspace elements, the memory deficit is 7,607,617 elements. Your options at this point are to upgrade to a version of CART with more workspace, or to specify limitations on the structure of the tree. We offer two methods to specify growing limitations.

Setting Limits Using Model Setup

The easiest method to limit the growth of a tree is to use the **Model Setup—Advanced** tab; Tree Size options. By default, CART sets the maximum values, based on the dataset size, to assure that they can never be reached. Reducing these values will considerably reduce the amount of required workspace.



The image shows a dialog box titled "Tree Size". It contains two settings: "Maximum number of nodes:" with a value of 8192, and "Depth:" with a value of 20. Both values are displayed in a text box with a small up/down arrow icon to its right.

We suggest, however, that you use caution when reducing these limits. The initial objective should be to reduce these values without creating a shortfall for the maximal tree. As long as the maximal tree size is less than the limitation you have set, you need not be concerned that the "true" optimal tree (one grown without limitations) will be grown. It is only when the imposed limits prevent completing the tree-growing process so as to grow the maximal tree that concern should arise.

For example, if you set the "Maximum number of nodes: 5000," and the tree sequence indicates the maximal tree contains 1500 nodes, you can clearly see that the maximal tree was grown without limitation. However, if you set the "Maximum number of nodes: 1000" and the tree sequence indicates the maximal tree contains 985 nodes, you may suspect that the maximal tree was never attained. When this occurs, the Tree Sequence report, found in the CART Report window, will be followed by a message that reads "Limited tree produced, complexity values may be erroneous."

Maximum number of nodes

Forces the tree generation process to stop when a specified number of nodes (both internal plus terminal) are produced.

Depth

Forces the tree generation process to stop after a specified tree depth is reached. The root node corresponds to the depth of 0.



Command-line users will use the following command syntax.

```
LIMIT NODES = <N>, DEPTH = <N>
```

Setting Limits Using Model Setup-Advanced tab

Alternative methods to limit the growth of a tree can be found in the **Model Setup-Advanced** tab. We are displaying the relevant portions of the Advanced tab as follows:

The screenshot shows a software interface with two sections. The top section, titled 'Tree Size', contains two settings: 'Maximum number of nodes:' set to 'AUTO' and 'Depth:' set to 'AUTO'. The bottom section, titled 'Sample Sizes', contains three settings: 'Learn Sample Size:' set to '293', 'Test Sample Size:' set to '293', and 'Subsample Size:' set to '293'. Each setting is displayed next to a text box with up and down arrow controls.

The parameter table displayed in the middle panel is a guide to tailoring the problem to the available resources. The easily adjustable parameters listed in the first column of the table are defined below:

Maximum Nodes	Forecast of the number of terminal nodes in the maximal tree
Depth	Forecast of the depth of the maximal tree
Learn Sample Size	Number of cases in the learn data set
Test Sample Size	Number of cases in the test data set
Sub-Sampling	Node size above which a random sub-sample (v. the full sample) is used to locate splits (default=learn sample size)

To manually set any one parameter individually (or any combination), enter a value into the corresponding text box.



You can save the values entered in the **Model Setup—Advanced** tab by clicking the [Defaults] button.

Report Writer

CART includes Report Writer, a report generator, word processor and text editor that allows you to construct custom reports from results diagrams, tables and graphs as well as the “classic” CART output appearing in the **Classic Output** window.

Using the Report Writer is easy! One way is to copy certain reports and diagrams to the **Report** window as you view the CART results dialog or output windows. Once processing is complete, a CART results window appears, allowing you to explore the performance with a variety of graphic reports, statistics, and diagrams. Virtually any graph, table, grid display, or diagram can be copied to the Report Writer. Simply right-click the item you wish to add to the Report Writer and select **Add to Report**. The selection will appear at the bottom of the Report window.

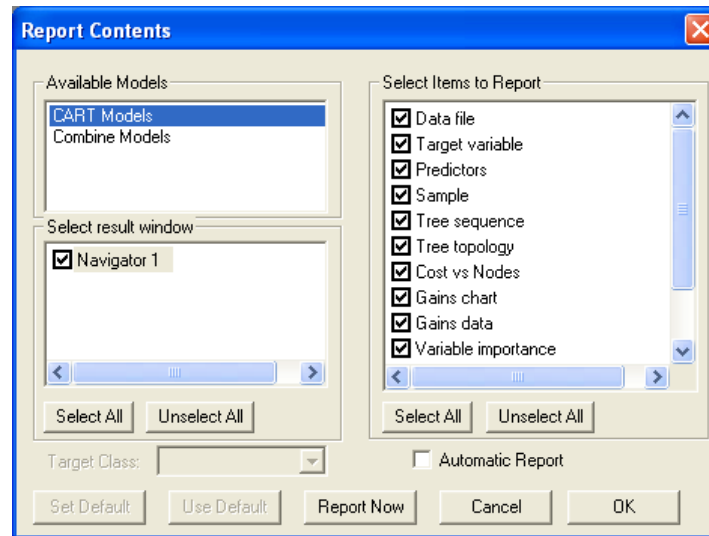
CART also produces “classic” output for those users more comfortable with a text-based summary of the model and its performance. To add any (or all) of CART’s classic output to the Report Writer window, highlight text in the classic output window, copy it to the Windows clipboard (**Ctrl+C**), switch to the Report Writer window and paste (**Ctrl+V**) at the point you want text inserted. Thus, you can combine those CART result elements you find most useful—either graphic in nature and originating in the CART results dialog, or textual in nature from the classic output - into a single custom report.

✂ Only one Report window is available at a time.

✂ To see whether a given table or chart can be added to the Report, simply right-click on the item you wish to add and see whether the **Add to Report** line is available in the pop-up menu. If it is available, click on it and the item will appear at the bottom of the Report window.

Default Options

In the **Report Contents** dialog, the currently-selected items to report and the Automatic Report checkbox can be saved as a default group of settings for future CART sessions by clicking the **[Set Default]** button. These default options will then persist from session to session because they are saved in the CART preference file (CART6.INI). You may recall these settings at any time with the **[Use Default]** button.



✎ CART 6 contains two sets of report options. One is for standard one-tree models, the other is for the combined bagging and ARCing models.

Additionally, CART can produce a “stock report” with the click of a button. You decide which components of the CART output would be most useful to you on the **Report—Set Report Options...** menu and then select them. The stock report will be the same for all CART results in the session until you visit the **Report Contents** dialog again. (In addition, the currently-open CART results dialogs are listed and individual ones can be excluded or added to the list that will appear in the report when the **[Report Now]** button is clicked.)

A stock report for the CART results that are currently active (i.e., in the foreground) can be generated by choosing **Report—Report Current**. If the active window is not a results window, the **Report Current** menu item will be disabled. Furthermore, if you have several CART results windows open, you can generate a report for all the trees (in the order in which they were built) by choosing the **Report—Report All** menu item.

Default Target Class

Reports summarizing class performance (e.g., gains charts) require a target class. For binary models (i.e., 0/1 or 1/2), the second level is assumed to be the target class. For multinomial models (e.g., 1, 2, 3, 4), the lowest class is assumed to be the target class.

Printing and Saving Reports

Once you have generated a report it may be printed or previewed by using the **Print...**, **Print Setup...** and **Print Preview...** options on the **F**ile menu.

To save a report to a file, use the **F**ile—**S**ave **A**s... option. The contents of the Report window can be saved in three formats: rich text format (.rtf), text, or text with line breaks (.txt). The rich text (.rtf) can be read by most other word processors and maintains the integrity of any graphics imbedded in the report. Neither text format retains graph or diagram images or table formatting.

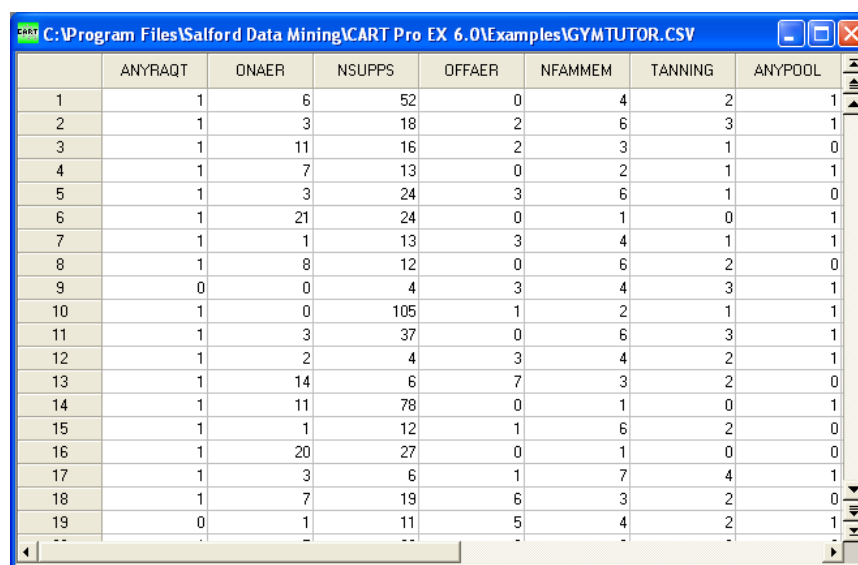
It is possible to cut and paste to/from the Report Window and other Windows documents, such as Microsoft Word, Notepad, Wordpad, etc. To select the entire report quickly and drop it into another Windows application, use **Ctrl+A** (shortcut for **E**dit -> **S**elect **A**ll), then **Ctrl+C** (copy to clipboard), move to the other application and paste.

Data Viewer

Once you have opened your data base, CART's Data Viewer allows you to view (but not edit or print) the data as a spreadsheet for investigating data anomalies or seeing the pattern of missing values.

The Data Viewer window is opened by selecting the **V**iew—**V**iew Data... menu item or clicking on the View Data toolbar icon .


 Only one data file can be displayed at a time.

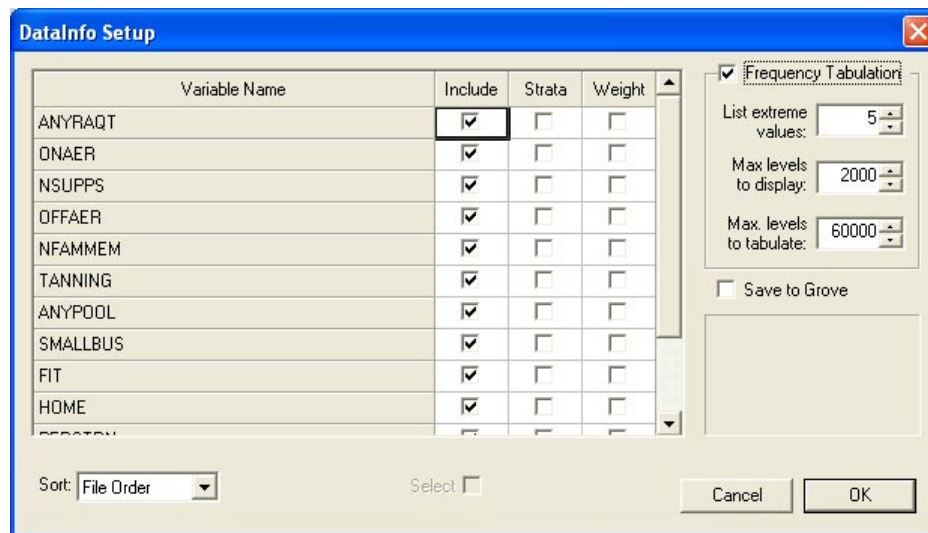


	ANYRAQT	ONAER	NSUPPS	OFFAER	NFAMMEM	TANNING	ANYPPOOL
1	1	6	52	0	4	2	1
2	1	3	18	2	6	3	1
3	1	11	16	2	3	1	0
4	1	7	13	0	2	1	1
5	1	3	24	3	6	1	0
6	1	21	24	0	1	0	1
7	1	1	13	3	4	1	1
8	1	8	12	0	6	2	0
9	0	0	4	3	4	3	1
10	1	0	105	1	2	1	1
11	1	3	37	0	6	3	1
12	1	2	4	3	4	2	1
13	1	14	6	7	3	2	0
14	1	11	78	0	1	0	1
15	1	1	12	1	6	2	0
16	1	20	27	0	1	0	0
17	1	3	6	1	7	4	1
18	1	7	19	6	3	2	0
19	0	1	11	5	4	2	1
--	.	.	--

Data Information

CART provides a GUI facility for viewing information on the currently-open data file. Information is provided in groups of descriptive statistics for each variable (numeric and character).

The **DataInfo Setup** window is opened by selecting the **View→Data Info...** menu item, or by clicking the  toolbar icon. This action will open the **DataInfo Setup** dialog. Here you can see various details about the data information that will be generated. It appears as follows.



Include	Select the variables to include and place a checkmark in the Include column
Strata	Define a single stratification variable for data information statistics by placing a checkmark in the Strata column (max. of eight levels)
Weight	Define a single weighting variable by placing a checkmark in the Weight column
Frequency Tabulation	Enable frequency tabulations
List Extreme Values	Specify the number of most- and least-frequent levels for display
Levels to Display	Specify the maximum number of discrete levels to display
Levels to Tabulate	Specify the maximum number of discrete levels to track
Save to Grove	Specify a grove file where data information results are saved

After you have made your selections using the **DataInfo Setup** dialog, click the **[OK]** button to proceed with the processing.

Once the resulting window is open and active, you will see two different views from which you can select by using the **[Brief]** and **[Full]** buttons.

The “Brief” view provides a snapshot of the data, including the number of records, number of missing values, percent missing, number of distinct levels, mean, minimum, and maximum values. The following is an example of this view.

Variable	N	N Missing	% Missing	N Distinct	Mean	Min	Max
ANYRAQT	293	0	0	2	0.27986	0	1
ONAER	293	0	0	20	4.0307	0	30
NSUPPS	293	0	0	53	16.324	0	105
OFFAER	293	0	0	12	1.4266	0	12
NFAMMEM	293	0	0	18	5.058	1	32
TANNING	293	0	0	7	1.5904	0	6
ANYPOOL	293	0	0	2	0.1843	0	1
SMALLBUS	293	0	0	2	0.051195	0	1
FIT	293	0	0	56	3.9564	0	10.127
HOME	293	0	0	2	0.037543	0	1
PERSTRN	293	0	0	1	0	0	0
CLASSES	293	0	0	4	0.50171	0	3
SEGMENT	293	0	0	3	2.0102	1	3

When the user clicks the **[Full]** button, more details can be seen about the data. Use the **[+]** and **[-]** toggles to expand and contract each information group. The information groups available for viewing include the following:

- DESCRIPTIVE:** N, N missing, N = 0, N <> 0, N Distinct Values, Mean, Std Deviation, Skewness, Coeff Variation, Cond. Mean, Sum of Weights, Sum, Variance, Kurtosis, Std Error Mean
- LOCATION:** Mean, Median, Range
- VARIABILITY:** Std Deviation, Variance, Intrqrt Range
- QUANTILES:** 100% Max, 99%, 95%, 90%, 75% Q3, 50% Median, 25% Q1, 10%, 5%, 1%, 0% Min
- FREQUENCY TABLES:** Most (Top 5 in Pop.), Least (Bottom 5 in Pop.), All

ANYRAQT		ONAER	
DESCRIPTIVE		DESCRIPTIVE	
N	293	N	293
N missing	0	N missing	0
N = 0	211	N = 0	38
N <> 0	82	N <> 0	255
N Distinct Values	2	N Distinct Values	20
Mean	0.27986	Mean	4.0307
Std Deviation	0.4497	Std Deviation	4.4123
Skewness	0.98071	Skewness	2.9375
Coeff Variation	1.6069	Coeff Variation	1.0947
Cond. Mean	1	Cond. Mean	4.6314
Sum of Weights	293	Sum of Weights	293
Sum	82	Sum	1181
Variance	0.20223	Variance	19.468
Kurtosis	-1.0382	Kurtosis	11.163
Std Error Mean	0.026272	Std Error Mean	0.25777
LOCATION		LOCATION	
VARIABILITY		VARIABILITY	
QUANTILES		QUANTILES	
FREQUENCY TABLES		FREQUENCY TABLES	

Command line users should issue the following command:

```
DATAINFO <var1>, <var2>, ...
```

- DATAINFO without arguments generates data information for all variables present in the data set.
- GUI users may request **Data Information** for any specific list of variables by issuing the DATAINFO command with the variable list at the command prompt. The **Data Information** window will now contain information on the specified variables only.
- Requesting DATAINFO on large datasets may result in long processing times. This is a result of an exhaustive attempt to generate frequency tables for all variables with the specified number of discrete levels.

Chapter

13



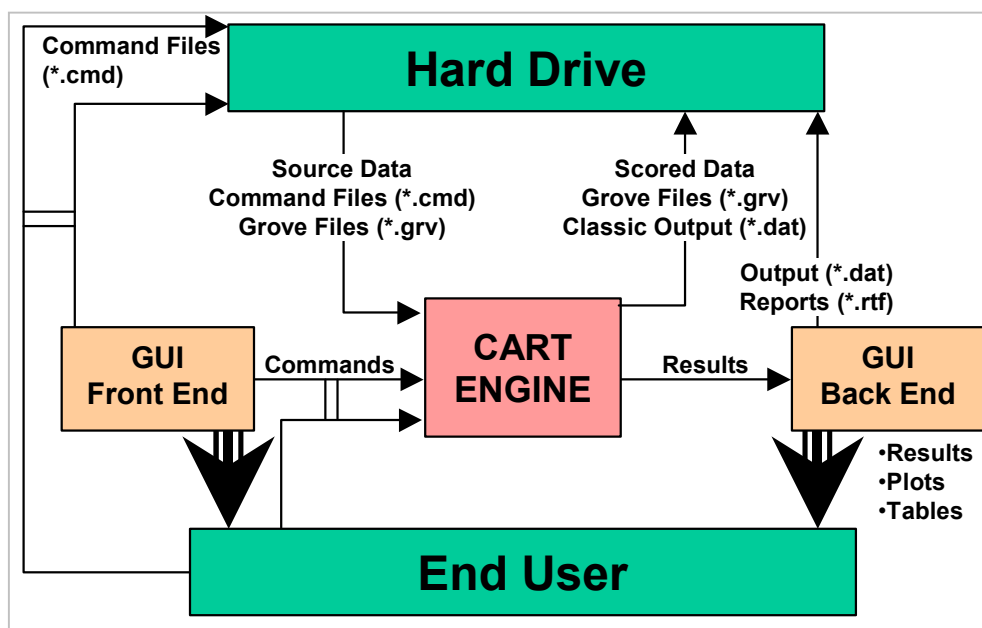
Working with Command Language

This chapter provides insight into the essentials of CART configuration and gives an important practical introduction to using command files.

Introduction to the Command Language

This chapter describes the situations in which a Windows user may want to take advantage of the two alternative modes of control in CART, command-line and batch, and provides a guide for using these two control modes. For users running CART on a UNIX platform, this chapter contains a detailed guide to command syntax and options and describes how the Windows version may assist you in learning the command-line language.

The following picture illustrates common channels of interaction between a user and CART.



First, note that CART itself is a sophisticated analytical engine controlled via command sequences sent to its input that can generate various pieces of output when requested.

An inexperienced user can communicate with the engine via the GUI front and back ends. The GUI front end provides a set of setup screens and “knows” how to issue the right command sequences according to the user’s input. It is also possible to request the GUI front end to save command sequences into an external command file. The GUI back end captures the results produced by the engine and displays various plots, tables, and reports. Most of these can be directly saved to the hard drive for future reference. The whole cycle (marked by the large arrows in the

diagram) is completely automated so that the user does not need to worry about what is taking place underneath.

A more demanding user may write separate command files with or without the help of the GUI front end. This feature is especially attractive for audit trail or various process automation tasks. Given that the current release of CART for UNIX is entirely command-line driven, the user running CART for UNIX will fall into this category.

The CART engine reads data off the hard drive for modeling or scoring, takes grove files for scoring, or executes command files when requested. In addition, the engine may generate new data with scoring information added, create grove files for models, and save classic text output.

The following sections provide in-depth discussions for users who have chosen to utilize command line controls.

Alternative Control Modes in CART for Windows

In addition to controlling CART with the graphical user interface (GUI), you can control the program via commands issued at the command prompt or via submission of a command (.cmd) file. This built-in flexibility enables you to avoid repetition, create an audit trail, and take advantage of the BASIC programming language.

Avoiding Repetition

You may need to interact with several dialogs to define your model and set model estimation options. This is particularly true when a model has a large number of variables or many categorical variables, or when more than just a few options must be set to build the desired model. Suppose that a series of runs are to be accomplished, with little variation between each. A batch command file, containing the commands that define the basic model and options, provides an easy way to perform many CART command functions in one user step. For each run in the series, the “core” batch command file can be submitted to CART, followed by the few graphical user interface selections necessary for the particular run in question.

Creating an Audit Trail

The Command Log window can help you create an audit trail when one is needed. Imagine not being able to reproduce a particular analysis track, perhaps because the specific set of options used to create a model (e.g., the name of the data set itself) was never recorded. The updated command log provides you with the entire command set necessary to exactly reproduce your analysis, provided the input data do not change.

Taking Advantage of CART's Built-In Programming Language

CART offers an integrated BASIC programming language that allows the user to define new variables, modify existing variables, access mathematical, statistical and probability distribution functions, and define flexible criteria to control case deletion and the partitioning of data into learn and test samples. BASIC commands are implemented through the command interface, either interactively or via batch command files.


Small BASIC programs are defined near the beginning of your analysis session, after you have opened your dataset but before you estimate (or apply) the model and usually before defining the list of predictor variables. BASIC is powerful enough that in many cases users do not need to resort to a stand-alone data manipulation program. See Appendix IV for more on the BASIC Programming Language.

Command-Line Mode

Choosing **Command Prompt** from the **File** menu allows you to enter commands directly from the keyboard. Switching to the command-line mode also enables you to access the integrated BASIC programming language. See Appendix IV for a detailed description of the BASIC programming language.

✎ This menu item is available only when the **CART Output** window is active. The command line prompt is marked by the ">" symbol and a vertical blinking cursor at the lower end of the right panel of the **CART Output** window.

Creating and Submitting Batch Files

The CART Notepad can be used to create and edit command files. From the Notepad, you can submit part or all of an open file. To submit a section of the command file, move the cursor to the first line of the selected section and select **Submit Current Line to End** from the **File** menu. To submit the entire command file, select **Submit Window** from the **File** menu (or click on the  in the toolbar). After you submit the file, the analysis proceeds as if you had clicked on the **[Start]** button in the GUI. The progress report window appears and, after the analysis is complete, the Results dialog is opened.

✎ These menu items are available only when the Notepad window is active (see below).

To submit an existing batch file, choose **Submit Command File** from the **File** menu. In the **Submit Command File** dialog that appears, specify the ASCII text file from which command input is to be read and then click on **[Open]**. To facilitate multiple

CART runs, the CART results are directed only to the CART Output window in text form (i.e., the GUI Results dialog does not appear).

✂ This menu item is available only when the **CART Output** window is active.

Each of these topics is discussed in more detail below.

Command Log

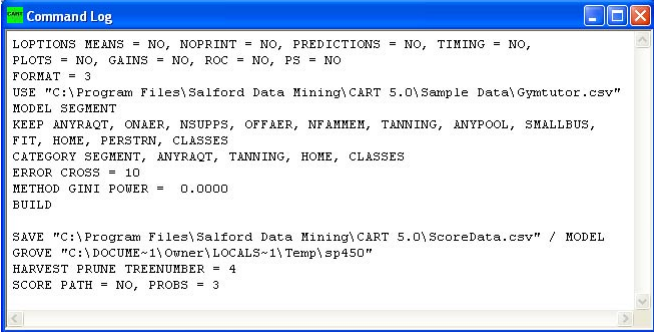
Most GUI dialog and menu selections have command analogs that are automatically sent to the Command Log and can be viewed, edited, resubmitted and saved via the Command Log window. When the command log is first opened (by selecting **Open Command Log...** from the **V**iew menu), all the commands for the current CART session are displayed. Subsequently, by selecting **Update Command Log** from the **V**iew menu, the most recent commands are added to the Command Log window.

✂ This menu item is available only when the Command Log window is active.

After computing a CART model, the entire set of commands can be archived by updating the command log, highlighting and copying the commands to the Notepad (or saving directly to a text file), then pasting them into your text application. Alternatively, you can edit the text commands, deleting or adding new commands, and then resubmit the analysis by selecting either **Submit Window** or **Submit Current Line to End** from the **F**ile menu.

View—Open Command Log

Within a single work session CART keeps a complete log of all the commands given to the engine. You may access this command list at any time through the **V**iew—**Open Command Log** menu.



```

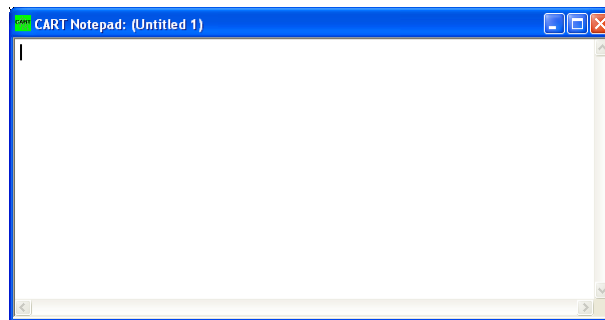
Command Log
LOPTIONS MEANS = NO, NOPRINT = NO, PREDICTIONS = NO, TIMING = NO,
PLOTS = NO, GAINS = NO, ROC = NO, PS = NO
FORMAT = 3
USE "C:\Program Files\Salford Data Mining\CART 5.0\Sample Data\Gymtutor.csv"
MODEL SEGMENT
KEEP ANYRAQT, ONAER, NSUPPS, OFFAER, NFANHEM, TANNING, ANYPOOL, SMALLBUS,
FIT, HOME, PERSTN, CLASSES
CATEGORY SEGMENT, ANYRAQT, TANNING, HOME, CLASSES
ERROR CROSS = 10
METHOD GINI POWER = 0.0000
BUILD

SAVE "C:\Program Files\Salford Data Mining\CART 5.0\ScoreData.csv" / MODEL
GROVE "C:\DOCUME~1\Owner\LOCALS~1\Temp\sp450"
HARVEST PRUNE TREENUMBER = 4
SCORE PATH = NO, PROBS = 3
  
```

- ✎ This feature is helpful for learning command syntax and writing your own command files. All you need to do is set up run options using the GUI front end and then read the corresponding command sequence from the Command Log.
- ✎ You may save the Command Log into a command file on your hard drive using the **File->Save** menu. If you do this before exiting a CART session, the resulting command file will contain the audit trail of the entire session.
- ✎ The Command Log Window supports the cut-and-paste technique.

File—New Notepad

The CART GUI offers a simple text editor to write your own command files. You may open multiple instances of the Notepad window using the **File->New Notepad...** menu. You may also open an existing command file using the **File->Open->Command File...** menu.



- ✎ You may use the cut-and-paste technique to grab command sequences from the Command Log Window to edit in the notepad window.

File—Submit Window

This menu item allows you to submit a command sequence from a CART Notepad window to the CART engine. Using this channel does not suppress the results window generated by the GUI back end.

- ✎ This option is also available for the Command Log Window, in which case the entire session will be reproduced.

- Submitting multiple runs may produce too many open windows, seriously affecting your system's performance. Saving the contents of the notepad window into a command file and then using the **File->Submit Command File...** menu item (see the following section) may be preferable.

File—Submit Command File

This menu item allows you to submit a command file (*.cmd) directly to the CART engine. When this channel is used, all output sent to the GUI back end is completely suppressed.

- Use this mode when you want to execute multiple runs without cluttering the GUI with multiple results windows (which may slow things down and drag the system to a halt).
- Consider using the OUTPUT command to save the classic text result to an ASCII text file.
- Consider using the GROVE command to save the GUI results.

Command Syntax Conventions

CART command syntax follows the following conventions:

- Commands are case insensitive.
- Each command takes one line starting with a reserved keyword.
- A command may be split over multiple lines using a comma “,” as the line continuation character.
- No line may exceed 256 characters.

Example: A sample classification run

The contents of a CLASS.CMD sample command file is shown below. Line-by-line descriptions and comments follow.

```

1>> REM SAMPLE CLASSIFICATIN RUN
2>> REM*****
3>> REM =INPUT/OUTPUT FILES=
4>> REM*****
5>> USE "sample.csv"
6>> GROVE "class.grv"
7>> OUTPUT "class.dat"
8>> REM*****
9>> REM =OPTIONS SETTINGS=
10>> REM*****
11>> BOPTIONS SURROGATES=5 PRINT=5, COMPETITORS=5, TREELIST=10,
12>> BRIEF, SERULE=0, IMPORTANCE=1, MISSING=NO
13>> LOPTIONS MEANS=YES, NOPRINT=NO, PREDICTIONS=YES/BOTH,
14>> TIMING=YES, PLOTS=YES, GAINS=NO, ROC=NO
15>> FORMAT=7/UNDERFLOW
16>> LIMIT MINCHILD=100, ATOM=200, NODES=5000, DEPTH=50,
17>> LEARN=100000, TEST=100000, SUBSAMPLE=100000
18>> REM*****
19>> REM =MODEL SETUP=
20>> REM*****
21>> MODEL Y2
22>> CATEGORY Y2
23>> PRIORS SPECIFY -1 = .5, 1 = .5
24>> Misclassify Cost = 2 Classify 1 as -1
25>> Misclassify Cost = 3 Classify -1 as 1
26>> KEEP Z1$, Z2$, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10
27>> ERROR SEPAR = T
28>> METHOD GINI POWER = 0
29>> WEIGHT W
30>> PENALTY / MISSING = 1, 1, HLC = 1, 1
31>> REM*****
32>> REM =BUILD MODEL=
33>> REM*****
34>> BUILD
35>> REM*****
36>> REM =QUIT CART=
37>> REM*****
38>> QUIT
  
```

All lines starting with **REM** are comments and will be ignored by the command parser.

We have marked commands of special interest with **RED** numbers.

Commands 1 through 3 control which files will be used or created.

- 1>>** The **USE** command specifies the data set to be used in modeling.
- ♦ CART has built-in support for comma-separated ASCII files. You may also access other supported file formats using **DATABASE CONVERSION** drivers.
- ☞ Use the GUI Command Log facility to learn quickly how to access various available file formats through **DATABASE CONVERSION**.
- 2>>** The **GROVE** command specifies the binary grove file to be created in the current directory. This file, which contains detailed model information, will be needed for the scoring and translating described later.
- ♦ This binary file is needed to view trees and model results from inside the CART GUI. It includes complete information about the model-building process, including pruning sequences and multiple collections of trees when applicable.
- 3>>** The **OUTPUT** command specifies the classic output file. This text file will report basic information about the data, the model-building process, and the optimal tree. The contents of this file, which are somewhat limited, may be controlled using **LOPTIONS** and **FORMAT** commands.

Commands 4 through 7 control various engine settings.

- 4>>** The **BOPTIONS** command sets important model-building options.
- 5>>** The **LOPTIONS** command sets various reporting options.
- 6>>** The **FORMAT** command sets the number of decimal digits to be reported.
- 7>>** The **LIMIT** command sets various limits, including how many observations and variables are allowed, the largest tree size allowed, the largest tree depth, the smallest node size allowed, and whether sub-sampling will be used.

For the most part, the above commands should be left unchanged unless you need fine control over the CART engine. A more detailed description can be found in the Appendix III Command Reference.

Commands 8 through 16 specify model settings that usually change from run to run.

- 8>>** The **MODEL** command sets the target variable.
- 9>>** The **CATEGORY** command lists all categorical numeric variables.
- ♦ Character variables are always treated as categorical and need not be listed here.
 - ♦ For classification models, numeric targets must be declared categorical.
- 10>>** The **PRIORS** command sets the prior probabilities for all target classes.
- ♦ The commands **PRIORS DATA** or **PRIORS EQUAL** are useful aliases for common situations.
- 11>>** The **MISCLASSIFY** commands set the cost matrix.
- ☞ Only non-unit costs need to be introduced explicitly.

✂ There will be as many MISCLASSIFY commands as there are non-unit cost cells in the cost matrix.

12>> The **KEEP** command sets the predictor list.

✂ This command is NOT cumulative.

13>> The **ERROR** command specifies the LEARN/TEST partition method.

- ◆ In this example, a dummy variable T separates the TEST part (T=1) from the LEARN part (T=0). Other useful methods are PROP=<ratio> (proportion selected at random), FILE=<file> (test set in a separate file), and EXPLORE (do not proceed with testing).

14>> The **METHOD** command sets the improvement calculation method.

- ◆ The commands METHOD GINI and METHOD TWOING are the most widely-used methods.
- ◆ POWER>0 results in more even splits.

15>> The **WEIGHT** command sets the weight variable if applicable.

16>> The **PENALTY** command induces additional penalties on missing-value and high-level categorical predictors.

- ◆ For backwards compatibility with earlier CART engines, one should use the following command instead:

```
PENALTY / MISSING = 1, 0, HLC = 1, 0
```

The remaining two commands are “action” commands.

17>> The **BUILD** command signals the CART engine to start the model-building process.

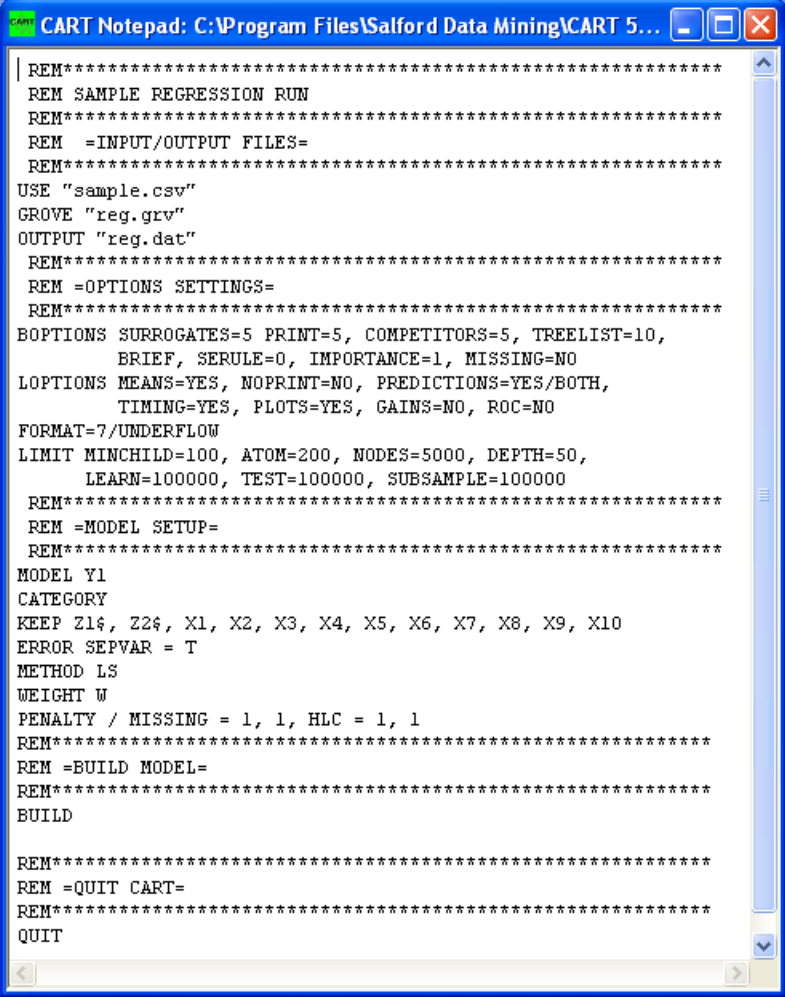
18>> The **QUIT** command terminates the program.

🔇 Anything following QUIT in the command file will be ignored.

Multiple runs may be conducted using a single command file by inserting additional commands.

Example: A sample regression run

The contents of a REG.CMD sample command file are shown below. Line-by-line descriptions and comments follow.



```

CART Notepad: C:\Program Files\Salford Data Mining\CART 5...
| REM*****
| REM SAMPLE REGRESSION RUN
| REM*****
| REM  =INPUT/OUTPUT FILES=
| REM*****
1>> USE "sample.csv"
2>> GROVE "reg.grv"
3>> OUTPUT "reg.dat"
| REM*****
| REM =OPTIONS SETTINGS=
| REM*****
4>> BOPTIONS SURROGATES=5 PRINT=5, COMPETITORS=5, TREELIST=10,
| BRIEF, SERULE=0, IMPORTANCE=1, MISSING=NO
5>> LOPTIONS MEANS=YES, NOPRINT=NO, PREDICTIONS=YES/BOTH,
| TIMING=YES, PLOTS=YES, GAINS=NO, ROC=NO
6>> FORMAT=7/UNDERFLOW
7>> LIMIT MINCHILD=100, ATOM=200, NODES=5000, DEPTH=50,
| LEARN=100000, TEST=100000, SUBSAMPLE=100000
| REM*****
| REM =MODEL SETUP=
| REM*****
8>> MODEL Y1
9>> CATEGORY
10>> KEEP Z14, Z24, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10
11>> ERROR SEPVAR = T
12>> METHOD LS
13>> WEIGHT W
14>> PENALTY / MISSING = 1, 1, HLC = 1, 1
| REM*****
| REM =BUILD MODEL=
| REM*****
15>> BUILD
| REM*****
16>> REM =QUIT CART=
| REM*****
| QUIT

```

All lines starting with **REM** are comments and will be ignored by the command parser.

We have marked commands of special interest with **RED** numbers.

If you have already mastered the classification run described in the previous section, note that the only differences are:

- ♦ The requested output file names have been changed in lines 2 and 3.
- ♦ The MODEL command (line 8) now uses a continuous target.
- ♦ The CATEGORY command (line 9) no longer lists our target.
- ♦ The PRIORS and Misclassify commands are no longer needed.
- ♦ The METHOD is changed to LS (least squares, line 12).

A detailed description of each command in this command file is provided below.

Commands 1 through 3 control which files will be used or created during this run.

- 1>>** The USE command specifies the data set to be used in modeling.
- ♦ CART has built-in support for comma-separated ASCII files
 - ♦ The **GROVE** command specifies the binary grove file to be created in the current directory. This file will contain detailed model information and will be needed for the scoring and translating described later.
 - ♦ This binary file is needed to view trees and model results from inside the CART GUI. It includes complete information about the model-building process, including pruning sequences and multiple collections of trees when applicable.
- 2>>** The **OUTPUT** command specifies the classic output file. This text file will report basic information about the data, the model-building process, and the optimal tree. The content of this file, which is controlled using the LOPTIONS and FORMAT commands, is somewhat limited.

Commands 4 through 7 control various engine settings.

- 3>>** The **BOPTIONS** command sets important model-building options.
- 4>>** The **LOPTIONS** command sets various reporting options.
- 5>>** The **FORMAT** command sets the number of decimal digits to be reported.
- 6>>** The **LIMIT** command sets various limits, including how many data are allowed, the largest tree size allowed, the largest tree depth, the smallest node size allowed, and whether sub-sampling will be used.

For the most part, these commands should be left unchanged unless you need fine control over the CART engine. A more detailed description can be found in the Appendix III Command Reference.

Commands 8 through 16 specify model settings that usually change from run to run.

- 7>>** The **MODEL** command sets the target variable.
- 8>>** The **CATEGORY** command lists all categorical numeric variables.

✂ Character variables are always treated as categorical and need not be listed here.

✂ In regression runs, the target is always a continuous numeric variable.

9>> The **KEEP** command sets the predictor list.

✂ This command is NOT cumulative.

10>> The **ERROR** command specifies the LEARN/TEST partition method.

- ◆ In this example, a dummy variable T separates the TEST part (T=1) from the LEARN part (T=0). Other useful methods are PROP=<ratio> (proportion selected at random), FILE=<file> (test set in a separate file), and EXPLORE (do not proceed with testing).

11>> The **METHOD** command sets the loss function.

- ◆ LS – least squares loss
- ◆ LAD – least absolute deviation loss

12>> The **WEIGHT** command sets the weight variable if applicable.

13>> The **PENALTY** command induces additional penalties on missing-value and high-level categorical predictors.

✂ We recommend always using the listed penalties.

- ◆ For backwards compatibility with earlier CART engines, one should use the following command instead:

```
PENALTY / MISSING = 1, 0, HLC = 1, 0
```

The remaining two commands are “action” commands.

14>> The **BUILD** command signals the CART engine to start the model-building process.

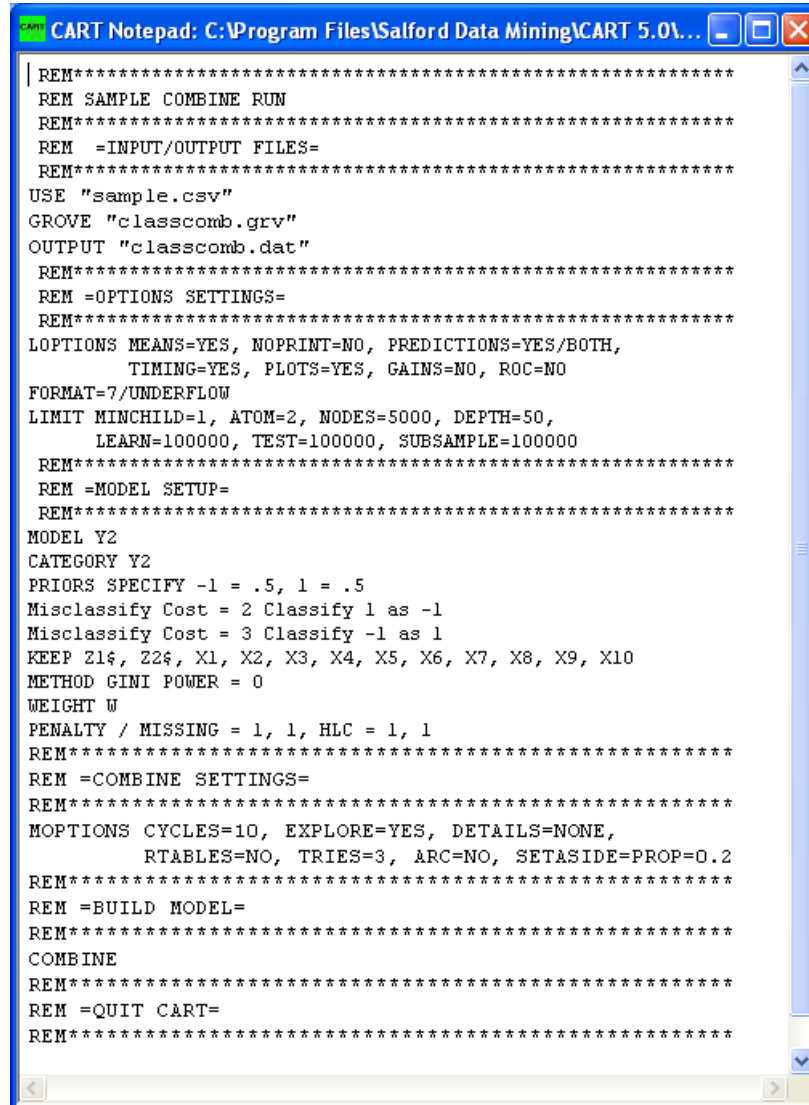
15>> The **QUIT** command terminates the program.

💡 Anything following QUIT in the command file will be ignored.

Multiple runs may be conducted using a single command file by inserting additional commands.

Example: Sample classification combine run

The contents of a CLASSCOMB.CMD sample command file are shown below. Line-by-line descriptions and comments follow.



```

REM*****
REM SAMPLE COMBINE RUN
REM*****
REM =INPUT/OUTPUT FILES=
REM*****
USE "sample.csv"
GROVE "classcomb.grv"
OUTPUT "classcomb.dat"
REM*****
REM =OPTIONS SETTINGS=
REM*****
LOPTIONS MEANS=YES, NOPRINT=NO, PREDICTIONS=YES/BOTH,
          TIMING=YES, PLOTS=YES, GAINS=NO, ROC=NO
FORMAT=7/UNDERFLOW
LIMIT MINCHILD=1, ATOM=2, NODES=5000, DEPTH=50,
      LEARN=100000, TEST=100000, SUBSAMPLE=100000
REM*****
REM =MODEL SETUP=
REM*****
MODEL Y2
CATEGORY Y2
PRIORS SPECIFY -1 = .5, 1 = .5
Misclassify Cost = 2 Classify 1 as -1
Misclassify Cost = 3 Classify -1 as 1
KEEP Z1$, Z2$, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10
METHOD GINI POWER = 0
WEIGHT W
PENALTY / MISSING = 1, 1, HLC = 1, 1
REM*****
REM =COMBINE SETTINGS=
REM*****
MOPTIONS CYCLES=10, EXPLORE=YES, DETAILS=NONE,
          RTABLES=NO, TRIES=3, ARC=NO, SETASIDE=PROP=0.2
REM*****
REM =BUILD MODEL=
REM*****
COMBINE
REM*****
REM =QUIT CART=
REM*****

```

This command file is almost identical with the CLASS.CMD command file (see earlier) with the following differences:

- ♦ Requested output file names have been changed in the OUTPUT and GROVE commands.
- ♦ The LIMIT settings have been changed to MINCHUILD=1, ATOM=2 in agreement with Leo Breiman's suggestions.
- ♦ The MOPTIONS command configures the combined run. See the Appendix III Command Reference for a complete description.

Example: Sample Scoring Run

The contents of a CLASSCOMB.CMD sample command file are shown below. Line-by-line descriptions and comments follow.

```

REM*****
REM SAMPLE SCORING RUN
REM*****
REM  =INPUT/OUTPUT FILES=
REM*****
1>> USE "gymtutor.csv"
2>> SAVE "ScoreOutGym.csv" / MODEL
3>> GROVE "gymtutor.GRV"
REM*****
REM  =SCORE MODEL=
REM*****
4>> SCORE DEPVAR=SEGMENT
REM*****
5>> REM  =QUIT CART=
REM*****
QUIT
  
```

A detailed description of each command in this command file is provided below.

Commands 1 through 3 control which files will be used or created during this run.

- 1>> The **USE** command specifies the data set to be used in modeling.
 - ♦ CART has built-in support for comma-separated ASCII files.
- 2>> The **SAVE** command specifies the case-by-case prediction output file. The specified file may contain case-by-case predictions, model variable values, path information, and class probabilities.

3>> The **GROVE** command specifies the binary grove file to be used for scoring.

Commands 4 through 5 control various engine settings.

4>> The **SCORE** command signals the CART engine to start the scoring process.

5>> The **QUIT** command terminates the program.

UNIX/Console Usage Notes

The nature of UNIX-like operating environments affects the operation of CART in non-trivial ways. This section discusses the operation of CART in the UNIX operating environment and the operation of console (non-GUI) CART in general. Both GUI and console CART are offered for Windows; only the console is offered for UNIX or Linux.

Case Sensitivity

CART's command interpreter is *case-insensitive*; in fact, commands are generally converted internally to upper-case letters (to include file names). The only exception to this rule is that text placed between quotation marks is not converted, remaining in its original case. UNIX file systems, on the other hand are *case-sensitive*, meaning that upper and lower case letters are treated as completely different characters. Thus, one could not refer to a file named "this.csv" as "THIS.CSV," or vice-versa.

It is therefore important to remember that unquoted filenames are assumed to be upper case; lower and mixed case names must be quoted.

Platform File Format Dependency

The Systat™ file format, traditionally used by CART, and other Salford Systems programs, is platform dependent. There are three known variations on the platforms we currently support:

- ♦ Big-endian UNIX (Solaris, IRIX, AIX, HP/UX)
- ♦ Little-endian UNIX (Alpha, Linux)
- ♦ DOS/Windows

The consequence of this is that Systat datasets created on Windows PCs cannot be read by CART under UNIX (and vice versa) unless the data translation engine is enabled (not currently available for AIX or IRIX). This is far less of a problem than it once was.

Use Caution When Transferring PC Files

It is always important to use binary mode when copying non-text files from a DOS/Windows environment to a UNIX environment (or vice-versa). Failure to do so **will** cause the files to be corrupted.

Supporting Database Conversion Libraries

On selected platforms, CART will use the Stat/Transfer database engine to read and write any file format supported by Stat/Transfer, provided that the interface is enabled. To access data through the Stat/Transfer interface, one simply uses the USE, SAVE, or ERROR FILE commands; the file name must be quoted, but no DBMS/COPY-style pseudo-extensions are required. To use the Stat/Transfer interface under Windows, the STATTRAN environment variable must point to the location of the Stat/Transfer libraries (not required under UNIX or Linux); to use the DBMS/COPY interface, the DBMSCOPY environment variable must point to the location of the DBMS/COPY libraries.

Beginning with CART 6, the Stat/Transfer interface, where present, takes precedence over the DBMS/COPY interface, which is disabled. To disable the Stat/Transfer interface, one can use the command "LOPTIONS STATTRAN=NO"; likewise, to re-enable the Stat/Transfer interface, one uses the command "LOPTIONS STATTRAN=YES." LOPTIONS DBMSCOPY can be similarly employed to enable or disable the DBMS/COPY interface. If both data translation engines are disabled, the only supported file formats are Systat and text.

✎ CART 6 includes native support for text datasets, which are, for many users, the most flexible and natural formats in which to maintain data. A single delimiter is used throughout the dataset. It is usually a comma, but semicolon, space, and tab are also supported as delimiters. (See Chapter 2: Reading Data; Reading ASCII files.)

The FPATH Command

The FPATH command can be used to specify locations for different types of input and output files. For example, the following command will cause CART to read and write files in the directory "Salford," under your home directory by default (on UNIX-like systems):

```
FPATH "~Salford"
```

Thereafter, if one gives an input/output command such as USE, OUTPUT, or SAVE, CART will look in ~/Salford unless the filename is quoted or the FPATH command is canceled by giving an FPATH command without arguments.

One can also specify different default directories for different sorts of files. To specify a default directory for input datasets, use:

```
fpath <pathname> /use
```

To specify a default directory for output datasets, use:

```
fpath <pathname> /save
```

For command files, use:

```
fpath <pathname> /submit
```

For text output files, use:

```
fpath <pathname> /output
```

FPATH without arguments restores the default, which is to use the current working directory. FPATH with an option but no pathname restores the default for the specified file type.

Online Help

Console CART has its own online help system, which can be accessed by opening CART in interactive mode and typing "HELP" at the prompt. To read the entry for a particular command, type "HELP," followed by the name of the command.

Workspace Allocation

Console CART can allocate arbitrary amounts of memory. The default workspace size is 25 MB, but this can be altered with either the SALFORD_M environment variable, or the -m command line flag. We suggest that SALFORD_M be set in the system-wide startup files (/etc/profile and /etc/csh.login on most UNIX-like systems), as appropriate for the hardware.

Limit on number of variables.

By default, CART will read datasets with up to 32,768 variables. This number can be increased with the -v command line flag.

Modes of Operation

Console CART can be invoked interactively by invoking it at the command prompt without arguments. You will get a series of startup messages looking something like this:

```
CART / TreeNet version 6.2.0.118
Copyright, 1991-2006, Salford Systems, San Diego, California, USA
Launched on 9/8/2006 with no expiration.
```

This launch supports up to 32768 variables.

```
Model space: 256 MB RAM allocated at launch, partitioned as:
  Real      : 65109998 cells
  Integer   : 1114112 cells
  Character: 3539016 cells
```

```
Data space, allocated as needed:
  The license supports up to 4096 MB of learn sample data.
```

```
Processing commands from: /usr/local/salford/lib/SALFORD.CMD
```

```
StatTransfer enabled.
```

```
>
```

You can then enter commands and get back responses. Your session ends when you enter the QUIT command. Since CART in interactive mode will accept commands through standard input and send responses through standard output, it is sometimes convenient to invoke it this way via a script or batch file.

Example: Read commands from a set of command files and write results to output.dat.

```
$ cat runit1.cmd runit2.cmd runit3.cmd | cart >output.dat
```

Generally, the more convenient way to run console CART is in batch mode, which can be invoked by specifying a command file as an argument.

Example: Execute runit1.cmd in batch mode.

```
$ cart runit1.cmd
```

When operating in batch mode, CART does not send any output to your screen, other than startup and error messages, unless ECHO ON is in effect, or the -e command line flag has been specified. It is therefore a good idea to specify an output file with the OUTPUT command inside your command file, otherwise you may never see the results at all. CART will terminate either when it has encountered a QUIT command, or there are no more commands to be executed.

Startup File

When console CART is started in interactive mode, it looks for a file named SALFORD.CMD, first in your current working directory and then in the directory pointed to by the SALFORD environment variable. If found, CART will execute its contents before displaying the command prompt. This allows one to specify default settings for all Salford Systems applications. SALFORD.CMD is not automatically executed in batch mode.

Command Line Startup Options

CART has a number of other command-line options, which can be shown by invoking CART with the -h flag:

 Command line syntax is:

```
cart [options] [commandfile] [options]
```

Options are:

e	Echo results to console
q	Quiet, suppress all output including errors
o<output_file>	Direct text results to a file
u<use_file>	Attach to a dataset
d<Path>	Identify DBMSCOPY dll path
w<Path>	Identify Stat/Transfer dll path (not required under UNIX)
t<Path>	Identify scratch file path
s<MBytes>	Data amount in MB, subject to license threshold
m<MBytes>	Model space in MB, subject to hardware limits
l<optional_logfile>	Error/warnings to text logfile
mt<N>	Max ternary size, 0 to grow tables without bound
v<N>	Specifies max N variables for the session

 Examples:

```
cart -e model1.cmd
cart /DataMining/Jobs-1/simulate.cmd -q
cart job1.cmd -o/RESULTS/job1.txt -u/AnalysisData/sample1.sys
cart -d/Progra-1/DBMSCopy7 -u/MyData/joint_data.xls[xls5]
cart -s512 -p64 -m128
```

 Environment variables can be used in lieu of command line switches:

SALFORD_S	in lieu of -s
SALFORD_M	in lieu of -m
SALFORD_P	in lieu of -p

Appendix I

Command Line Menu Equivalents

*This appendix provides an overview of
command line equivalents to the
graphical user interface options.*

Command	Pull-Down Menu [Dialog]
ADJUST	Limits–Growing Limits
AUXILIARY	Model–Construct Model [Model]
BATTERY	Model–Construct Model [Battery]
BOPTIONS	
SERULE	Model–Construct Model [Best Tree]
COMPLEXITY	Model–Construct Model [Advanced]
COMPETITORS	Model–Construct Model [Best Tree]
CPRINT	Edit–Options [CART]
TREELIST	Edit–Options [CART]
SPLITS	Command Line Only
SURROGATES	<u>How Many to Store:</u> Model–Construct Model [Best Tree] <u>How Many to Report (SURROGATES PRINT):</u> Edits–Options [CART]
SCALED	Model–Construct Model [Advanced]
NCLASSES	Model–Construct Model [Categorical]
CVLEARN	Model–Construct Model [Advanced]
PAGEBREAK	Command Line Only
NODEBREAK	Command Line Only
COPIOUS	Edit–Options [CART]
BRIEF	Edit–Options [CART]
OPTIONS	Command Line Only
IMPORTANCE	Command Line Only
QUICKPRUNE	Command Line Only
DIAGREPORT	Command Line Only
HLC	Command Line Only
PROGRESS	Command Line Only
MISSING	Model–Construct Model [Advanced]
MREPORT	Command Line Only
VARDEF	Command Line Only
CVS	Command Line Only
PLC	Command Line Only
BUILD	Model–Run CART
CATEGORY	Model–Construct Model [Model]
CDF	Command line only
CLASS	Model–Construct Model [Categorical], [Set Class Names]
COMBINE	Model–Construct Model [Combine]
DATAINFO	View–Data Info
DESCRIPTIVE	Command line only

ECHO	File—Log Results To
ERROR	Model—Construct Model [Testing]
EXCLUDE	Command line only
FORCE	Model—Construct Model [Force Split]
FORMAT	Edit—Options [Reporting]
FPATH	Command line only
GROVE	Model—Score Data Model—Translate Model
HARVEST	Command line only
HELP	Help—CART Help
HISTOGRAM	Command line only
IDVAR	Model—Score Data
KEEP	Model—Construct Model [Model]
LIMIT	Model—Construct Model [Advanced]
LINEAR	Model—Construct Model [Method]
EXHAUSTIVE	Command line only
LCLIST	Model—Construct Model [Method]
LOPTIONS	
MEANS	Edit—Options [General]
TIMING	Edit—Options [General]
NOPRINT	Edit—Options [CART]
PREDICTION_SUCCESS	Edit—Options [General]
GAINS	Edit—Options [General]
ROC	Edit—Options [General]
PS	Edit—Options [General]
UNS	Model—Construct Model [Model]
UNR	Command line only
PLOTS	<u>Whether to show</u> Edit—Options [CART] <u>Character to use</u> Command line only
DBMSCOPY	Command line only
STATTRAN	Command line only
MEMO	Command line only
MEMORY	Limits—Growing Limits
METHOD	Model—Construct Model [Method]
MISCLASS	Model—Construct Model [Costs]
MODEL	Model—Construct Model [Model]

MOPTIONS	Model–Construct Model [Combine]
CYCLES	number of trees to combine
SETASIDE	evaluation sample holdout method
TEST	pruning test method – Use resampling training data
CROSS	pruning test method – Cross-validation
EXPLORE	pruning test method – No pruning
ROOT	trees: root name
REMAP	combine method, arcing exponent
ARC	combine method
LROOT	learn sample: root name
DETAILS	report details: initial, committee
RTABLES	report details: repeated cases
NAMES	Model–Construct Model [Model]
NEW	File–Clear Workspace
NOTE	Command line only
OPTIONS	Command line only
OUTPUT	File–Save CART output
PAGE	Command line only
PARTITION	Model–Construct Model [Testing]
PENALTY	Model–Construct Model [Penalty]
PRIORS	Model–Construct Model [Priors]
PRINT	Command line only
QUIT	File–Exit or <Alt+F4>
REM	Command line only
SAVE	Model–Score Data
SCORE	Model–Score Data
STRATA	View–Data Info
SEED	Edit–Options [CART]
SELECT	Model–Construct Model [Select Cases]
SUBMIT	File–Submit Command File
TRANSLATE	Model–Translate Model
USE	File–Open–Data File
WEIGHT	Model–Construct Model [Model]
XYPLOT	Command line only

Appendix II

Errors and Warnings

*This appendix provides information
on common errors and warnings.*

If you have any difficulty understanding or resolving any of the following errors and warnings, please contact your technical support representative at Salford Systems.

Error #1: UNABLE TO UNDERSTAND WHAT YOU MEAN ABOUT HERE...

The program has encountered a problem with your command file syntax that it cannot resolve. Check the syntax immediately before and after the position indicated in the error message.

Error #2: YOU CANNOT WRITE TO A FILE YOU ARE READING FROM

You are attempting to use the same file for reading and writing. Check the USE and SAVE commands.

Also make sure that none of the files involved are currently open in another application.

Error #3: THE PROBLEM IS TOO LARGE FOR THIS VERSION.

CART does not have enough resources to complete your run. Check the run settings; certain extreme situations such as high-level categorical predictors and targets can render your run impossible to conduct.

Contact Salford Systems if this message appears under "normal" settings.

Error #4: INCORRECT FILE ASSIGNMENT. NO ASSIGNMENT MADE

The OS was not able to open your file. Check your USE, GROVE, and INCLUDE commands.

Also, make sure that none of the files involved is held by another application.

Error #5: ILLEGAL VALUES FOR SUBSCRIPT

Array variables are limited to 99 elements. Anything beyond that will trigger this error message.

Error #8: YOU ARE TRYING TO PROCESS THE WRONG KIND OF DATA

Check that your data file has the right format and is not corrupted.

Error #10: YOU ARE TRYING TO READ AN EMPTY OR NONEXISTENT FILE OR YOUR FILE IS IN A DIFFERENT DIRECTORY

CART is not able to open one of the files. Check your USE/GROVE/SUBMIT commands for possible errors.

Also, make sure that none of the files involved is held by another application.

Error #12: UNEXPECTED END OF FILE ENCOUNTERED

The file you are reading from is corrupt. Try another version of the same file or consider using another data format.

Error #13: YOU HAVE NOT GIVEN AN INPUT FILE WITH USE COMMAND

See the USE command in the command reference

Error #14: YOU CANNOT HAVE MORE THAN FIVE NESTED INCLUDE FILES

CART command parser allows no more than five nested INCLUDE statements. Consider rearranging your scripts into fewer layers.

Error #24: TEMPORARY FILE CREATE FAILED

CART creates temporary files in a dedicated folder needed for its work. Check that there is enough space in the temporary folder and that you have the write permission to that folder.

Error #28: Too many variables in your dataset

You have exceeded CART's limit on the number of variables (currently 8128). Note that new variables created by BASIC and missing value indicators are treated as legitimate variables and may cause the total number of predictors to go beyond the limit.

Error #10002: NO INDEPENDENT VARIABLES WERE SPECIFIED FOR THIS MODEL

Check your KEEP or EXCLUDE commands

Error #10005: THE NUMBER OF DEPENDENT VARIABLE CATEGORIES IS NOT EQUAL TO THE NUMBER OF PRIOR CLASS PROBABILITIES

Make sure that you have listed all available levels in the PRIORS SPECIFY command.

Error #10006: YOU HAVE SPECIFIED THE DEPENDENT VARIABLE AS THE SEPARATION VARIABLE

CART does not allow the use of the same variable in the MODEL and ERROR commands.

Error #10007: AN INDEPENDENT VARIABLE WAS SPECIFIED AS THE SEPARATION VARIABLE

CART does not allow the use of the same variable in the ERROR and KEEP (or EXCLUDE) commands.

Error #10008: DATASET HAS NO NUMERIC VARIABLES IN COMMON WITH YOUR "USE" DATASET

You are using the wrong test set.

Error #10009: MISCLASSIFICATION COSTS MUST BE POSITIVE AND NONZERO

Use small positive numbers (such as .001) to reflect zero costs

Error #10011: OUT OF MEMORY, SPLIT INTO SEVERAL SMALLER COMMANDS

The command parser has encountered difficulties processing one of your commands due to its length. Consider alternative ways to use smaller commands.

Error #10014: NOT ENOUGH MEMORY TO DISPLAY A MISCLASSIFICATION MATRIX

The prediction success tables cannot be displayed because you have too many distinct classes in your target.

Error #10015: YOU HAVE NOT SPECIFIED A TREE FILE YET

Check for the presence of the GROVE command in your scoring runs.

Error #10017: Unable to locate or open your GROVE file

Check the GROVE command. Make sure the grove file is not held by another application.

Error #10018: THE ABOVE VARIABLE IS PART OF THE TREE AND MUST BE PRESENT ON THE CASE-BY-CASE DATA SET

The file you are trying to score does not have one of the variables that were part of the model.

To enforce the scoring anyway, you must complete your file with all missing model variables with values set to missing.

Error #10019: Your grove file does not contain any CART trees

You are probably trying to use a grove file generated by TreeNet or MARS.

Check your GROVE command.

Error #10021: PRIORS SUM TO ZERO OR A NEGATIVE NUMBER

Check the PRIORS SPECIFY command - the priors cannot be negative numbers or all zeroes.

Error #10023: Unable to proceed with model estimation

CART has encountered a situation that prevents further modeling. Check your run settings and your data.

Error #10024: The CASE command has been replaced by the SCORE

Replace CASE with SCORE in your command file.

Error #10025: No learn sample variance in target variable

Your target has the same value for all learn records. Because it makes no sense to proceed, modeling ends.

Error #10026: THE ABOVE VARIABLE IS ONE OF THE INDEPENDENT VARIABLES OF THE TREE AND MAY NOT BE USED AS THE DEPENDENT VARIABLE

Check your MODEL and KEEP (or EXCLUDE) commands and make sure they do not overlap.

Error #10050: UNABLE TO LOAD ANY MORE DATA INTO RAM

Increase the amount of RAM available on your machine.

Error #10055: Too many redraws trying to construct ARC resampling

The ARC process has collapsed. Use "exploratory trees" to reduce the chance this error will occur.

Error #10057: The above variable name in the model KEEP list has an illegal leading character

Read the variable names requirements in the manual.

Error #10063: Error with prior class probabilities

Check the PRIORS command.

Error #10065: Not enough memory to add the missing value indicators that your data require

The total number of variables, including missing value indicators, exceeds the maximum allowed limit of 8128.

Error #10066: The center cut power exponent can be no larger than 10.0

Modify the POWER= setting in the METHOD command appropriately.

Error #10067: The model involves a missing value indicator automatically generated from the above variable. The above variable must be present on the case-by-case data set

Add the variable mentioned to the data, filling it with missing values if it is unknown.

Error #10069: Unable to open the grove file:

Check the GROVE command.

Error #10070: Unable to identify model (eg: tree/treenet/mars) in grove

You have a corrupted grove file, the wrong version of the file, or the wrong model selection criteria.

Error #10072: Error creating grove file

Check for enough disk space and/or permissions.

Error #10074: Not enough memory available to estimate model

CART does not have enough resources to complete your run. Check the run settings; certain extreme situations such as high-level categorical predictors and targets can render your run impossible to conduct.

Contact Salford Systems if this message appears under "normal" settings.

Error #10075: Invalid MODEL command options, was expecting ...

Check the MODEL command.

Error #1008: Target had no variation after LAD transformation

This usually happens when the LAD method is activated on binary targets; switch to LSD or classification.

Error #11004: TOO MANY CATEGORICAL OR LINEAR COMBINATION SPLITS. TRY USING THE COMMANDS: BOPTION SPLITS, LINEAR LINSPLITS

The number of categorical or linear combination splits has exceeded the initially reserved amounts. Increase the limits using the corresponding commands.

Error #11005: TREE IS GROWING TOO DEEP. TRY USING COMMAND: LIMIT DEPTH

The tree depth exceeds the default maximum value. Use the LIMIT DEPTH command to increase it.

Error #11006: TOO MANY CATEGORICAL COMPETITOR SPLITS

The number of categorical splits has exceeded the initially reserved amount. Increase the limit using the BOPTION SPLITS command.

Error #11008: COMPUTATIONAL INSTABILITY DUE TO LINEAR COMBINATIONS. TRY DISABLING LINEAR COMBINATIONS AND RERUN.

Contact Salford systems with details about your run.

Error #20008: YOU HAVE SPECIFIED MULTIPLE DEPENDENT VARIABLES...

Check the MODEL command - only one variable is allowed there.

Error #20011: COMPUTATIONAL DIFFICULTIES ENCOUNTERED, UNABLE TO CONTINUE

Contact Salford systems with details about your run.

Error #20068: Unable to discern a valid set of variable names from your text dataset

Make sure that the correct value separator is used and that the first line lists the variable names.

Error #20069: Unable to open your text dataset

Check the file location and USE command.

Error #20071: You have not specified a grove file yet

Add the GROVE command appropriately.

Error #20076: Error managing data swap file, cannot continue

Proceed with regular system maintenance; change swap file settings.

Warning #1: At least one variable had too many distinct values to tabulate completely. This is most likely to occur with character variables, especially those with long string values. Also, this may be due to treating a ordinal variable as discrete (categorical)...

Read carefully the entire warning and proceed with the recommendations.

Check the KEEP/EXCLUDE commands.

Warning #2: The following variables had more than 2000 distinct values...

Check the KEEP/EXCLUDE commands for the presence of undesirable predictors.

Warning #3: CART is using v-fold cross validation on a training sample with <N> records. Using a test sample will speed up the run

Your data set is large enough to allow a separate test set.

Warning #4: Singularity solving for linear combination split

CART has encountered difficulties finding linear combination splits - univariate splits will be used instead for the node where the difficulty appeared.

Warning #5: The optimal tree has no splits and one node...

According to the current set of PRIORS and COSTS the null tree is better than any other tree CART has grown.

This situation may also take place when growing regression trees on data sets with a lot of noise.

Warning #7: Obsolete syntax on CATEGORY command

CATEGORY command no longer requires explicit level counts in CART 6.

Warning #10: Case weights are not supported for linear combinations...

Support for weights in linear combinations will be implemented in future versions of CART.

Warning #11: Case weights are not supported for the LAD rule...

Support for weights in LAD regression will be implemented in future versions of CART.

Appendix III

Command Reference

This appendix provides a command language reference including syntax and examples.

ADJUST

Purpose

The **ADJUST** command facilitates resizing of critical memory management parameters.

The command syntax is:

```
ADJUST [ LEARN = <n>, TEST = <n>, ATOM = <n>, DEPTH = <n>,  
        NODES = <n>, SUBSAMPLE = <n> ]
```

All parameters entered but one should be followed by "**=<n>**" values. The one parameter on the ADJUST command NOT given a fixed value will be automatically adjusted to attempt to fit the problem into the available workspace.

Examples:

```
ADJUST ATOM=20, DEPTH=8, LEARN  
ADJUST LEARN=500, NODES  
ADJUST DEPTH
```

AUXILIARY

Purpose

The **AUXILIARY** command specifies variables (either in the model or not) for which node-specific statistics are to be computed. For continuous variables, statistics such as N, mean, min, max, sum, SD and percent missing may be computed. Which statistics are actually computed is specified with the DESCRIPTIVE command. For discrete/categorical variables, frequency tables are produced showing the most prevalent seven categories.

The command syntax is:

```
AUXILIARY <variable>, <variable>, ...
```

Examples:

```
AUXILIARY ONAER, NSUPPS, OFFAER
```

Variable groups may be used in the AUXILIARY command similarly to variable names.

BATTERY

Purpose

Results are saved into the grove file. The **BATTERY** command generates a group of models by varying one or more features or control parameters of the model. It is given prior to the BUILD command, which begins the model-building process. The various forms of the BATTERY command are:

BATTERY ATOM

Eight models are generated using ATOM values of 2, 5, 10, 25, 50, 100, 200 and 500.

BATTERY CV

Cross-validation trees, using 5, 10, 20 and 50 CV bins.

BATTERY DEPTH

Generates one unconstrained and seven depth-limited (1, 2, 3, 5, 10, 20, 50) models.

BATTERY FLIP

Generates two models, reversing the learn / test samples.

BATTERY MVI

Generates five models: main effects, main effects with MVIs (Missing value indicators), MVIs only, main effects with missing values penalized, main effects and MVIs with missing values penalized.

BATTERY MINCHILD

Eight models using minchild settings of 1, 2, 5, 10, 25, 50, 100 and 200.

BATTERY NEST [= YES | NO]

CART EX Pro only. Do we nest (combine) battery specifications or not? The default is no.

BATTERY NODES

CART EX Pro only. Four models, each limiting the number of nodes in a tree (4, 8, 16 and 32 terminal nodes).

BATTERY ONEOFF

CART EX Pro only. Attempt to model the target as a function of one predictor at a time. Note that for CART classification models, the class probability splitting rule is used.

BATTERY LOVO

CART EX Pro only. Repeat the model leaving one predictor out of the model each time. Note that for CART classification models, the class probability splitting rule is used, (the reverse of ONEOFF.)

BATTERY PRIOR = <target_class>

CART EX Pro only. Vary the priors for the specified class from 0.02 to 0.98 in steps of 0.02, i.e, 49 models. If you wish to specify a particular set of values, use the START, END and INCREMENT options, e.g.

```
BATTERY PRIOR=3 START=.5 (will infer END and INCREMENT settings)
BATTERY PRIOR="Male" START=.45, END=.75, INCREMENT=.01
```

BATTERY RULES

Generate a model for each splitting rule (six for classification, two for regression). Note that for the TWOING model, POWER is set to 1.0 to help ensure it differs from the GINI model.

BATTERY SHAVING [=<n>, TOP|BOTTOM|ERROR, STEPS=<n>]

CART EX Pro only. Shave predictors from the model, cycling until the specified number of steps have been completed (STEPS=) or until there are no predictors left. Can shave from the TOP (most important are shaved first) or BOTTOM. ERROR will build a full set of models before determining which single predictor can be best eliminated based on model error (not importance), repeating for each predictor that is shaved. TOP and BOTTOM can shave N at a time. The defaults are to shave one predictor at a time from the bottom until the model degenerates to nothing. Note that ERROR will proceed until the model degenerates, i.e., the STEPS option has no effect with ERROR.

BATTERY TARGET [MP=<yes|no>, MT=<yes|no>, MS=<yes|no>, SAVE=<"filename">]

CART EX Pro only. Attempt to model each variable in the KEEP list as a target, using all other variables in the KEEP list as predictors. MP governs whether MVIs (Missing value indicators) are used as predictors. MT governs whether MVIs are used as targets. MS governs whether MVIs are saved to the output dataset. SAVE saves the imputed values to a new dataset. If you wish to specify a list of targets separately from the KEEP list of predictors, use the syntax:

BATTERY TARGET=<target1>,<target2>,...

In this instance, variables can be part of both the TARGET list and KEEP but in the most common use the two lists would be mutually exclusive.

BATTERY CVR=<n>

Repeats the CV process N times with a different random seed each time.

BATTERY KEEP=<NK,NR> [CORE=<predictor>,<predictor>,...]

CART EX Pro only. Repeat the model NR times, selecting a subset of NK predictors from the KEEP list each time. The CORE option defines a group of predictors (from the main KEEP list) that are included in each of the models of the battery.

BATTERY MCT=<n>

Monte Carlo shuffling of the target. First model is unperturbed. Successive models have target shuffled to break the correlation between target and explanatory variables. MCT may only be run alone, or with RULES, in which case it will be nested.

BATTERY QUIET [=YES|NO|AUTO]

Some results that would be produced for a single model are not produced for certain batteries. You can disable this output for all batteries with BATTERY QUIET=YES, produce it with BATTERY QUIET=NO or allow the program to decide what output is presented with BATTERY QUIET=AUTO.

BATTERY PROXIMITY [=YES|NO]

Indicates whether a proximity matrix report should be produced for the battery. By default, it is produced for BATTERY TARGET only, but it is possible to produce this report for all batteries.

BATTERY PF=<"filename">

Saves the proximity matrix to a text (comma-separated) file.

BATTERY SAMPLE

Will result in a series of five models in which the learn sample is reduced randomly four times to examine the effect of learn sample size on error rate.

BATTERY DRAW=<proportion>,<nreps>

CART EX Pro only. Runs a series of models in which the learn sample is repeatedly drawn (without replacement) from the "main" learn sample. The test sample is not altered. The proportion to be drawn (in the range 0 to 1 exclusive) and the number of repetitions are specified, e.g.:

```
BATTERY DRAW=0.25,20
```

will repeat the model 20 times, each with a random 25% draw of the available learning data.

BATTERY SUB-SAMPLE

Varies the sample size that is used at each node to determine competitor and surrogate splits. The default values used are 100, 250, 500, 1000, 5000 and no sub-sampling. You may list a set of values with the VALUES option as well as a repetition factor (each sub-sampling size is repeated N times with a different random seed each time), e.g.:

```
BATTERY SUB-SAMPLE VALUES=1000,2000,5000,10000,20000,0  
BATTERY SUB-SAMPLE VALUES=1000,2000 REPEAT=20
```

In the above example, note that 0 indicates sub-sampling should not be used.

BOPTIONS

Purpose

The **BOPTIONS** command allows several advanced parameters to be set.

The command syntax is:

```
BOPTIONS  SERULE=<x>, COMPLEXITY=<x>, COMPETITORS=<n>, CPRINT=<n>,
           SPLITS=<n | AUTO>, SURROGATES=<n1> [PRINT=<n2>], OPTIONS,
           NCLASSES=<n>, CVLEARN=<n>, NOTEST, ECHO, TREELIST=<n>,
           PAGEBREAK=<"page_break_string">,
           NODEBREAK=<ALL | EVEN | ODD | NONE | <N>>,
           IMPORTANCE=<x>, COPIOUS | BRIEF, SCALED,
           QUICKPRUNE=<YES | NO>, DIAGREPORT=<YES | NO>,
           HLC=<n1>, <n2>, PLC=<YES | NO>, CVS=<YES | NO>,
           PROGRESS=<SHORT | LONG | NONE>,
           MISSING=<YES|NO|DISCRETE|CONTINUOUS|LIST=varlist>,
           MREPORT=<YES | NO>, VARDEF=<N | I>
```

in which <x> is a fractional or whole number and <n> is a whole number.

SERULE	The number of standard errors to be used in the optimal tree selection rule. The default is 0.0.
COMPLEXITY	Parameter limiting tree growth by penalizing complex trees. The default is 0.0—no penalty, trees grow unlimited.
COMPETITORS	Number of competing splits reported for each node. Default=5.
CPRINT	Number of competing splits printed for each node in the classic (text) output. Defaults to the COMPETITORS option.
TREELIST	Number of trees reported in tree sequence summary. Default=10.
SPLITS	Forecast of the number of splits (primary and surrogate) on categorical variables in maximal tree. This value is automatically estimated by CART but may be overridden.
SURROGATES	<n1> is the maximum number of surrogates to store in the binary tree file and to compute variable importance. Default= 5. <n2> is the number of surrogates to report for each node, and is set equal to <n1> if not specified.

SCALED	Indicates the complexity specified IS NOT relative. Any complexity specified as greater than 1.0 is considered scaled and the SCALED option is not required.
NCLASSES	For classification problems in which the number of dependent levels is greater than two, NCLASSES specifies the maximum number of classes allowed for an independent categorical variable for an exhaustive split search. For independent categorical variables with more levels, special "high-level categorical" algorithms are used (see the HLC option). Depending on the platform, for classification problems NCLASSES greater than 10-20 can result in significant increases in compute time. The default is 12. NOTE: For BINARY classification trees, special algorithms are used that allow exhaustive split searches for high-level categoricals with essentially no compute-time penalty.
CVLEARN	Sets the maximum number of cases allowed in the learning sample before cross validation is disallowed and a test sample required. The default is 3000.
PAGEBREAK	Defines a string that may be used to mark page breaks for later processing of CART text output. The page break string may be up to 96 characters long, and will be inserted before the tree sequence, the terminal node report, learn/test tables, variable importance and the final options listing. Page breaks are also inserted in the node detail output, according to the NODEBREAK options (see below). If the pagebreak string is blank, no pagebreaks are inserted.
NODEBREAK	This option is only active if you have defined a nonblank pagebreak string with the PAGEBREAK option. NODEBREAK allows you to specify how often the node detail report is broken by page breaks. The options are ALL, EVEN, ODD, NONE or you may specify a number (such as 3 or 10). The default is ODD, breaking prior to node 3, 5, etc. Even if you request NONE, there will still be a pagebreak prior to the node detail title.
COPIOUS BRIEF	COPIOUS reports detailed node information for all maximal trees grown in cross validation. The default is BRIEF.
OPTIONS	Provides a report of advanced control parameters at the end of tree building.

IMPORTANCE	Places weight on surrogate improvements when calculating variable importance. Must be between 0 and 1. The default is 1.0.
QUICKPRUNE	Invokes an algorithm that avoids rebuilding the tree after pruning has selected an optimally-sized tree.
DIAGREPORT	Produces tree diagnostic reports.
HLC	<p>Accommodates high cardinality categoricals. Assume the variable in question has <i>nlev</i> levels:</p> <ul style="list-style-type: none"><i>n1</i>: number of initial random split trials. <i><n1></i> must be greater than 0.<i>n2</i>: number of refinement passes. Each pass involves <i>nlev</i> trials. <i><n2></i> must be greater than 0. <p>The default is HCC=200,10. The HCC option is identical to HLC</p>
PROGRESS	Issues a progress report as the initial tree is built. This option is especially useful for trees that are slow to grow. LONG produces full information about the node, SHORT produces just the main splitter info, and NONE turns this feature off. The default is NONE.
MREPORT	Produces a special report summarizing the amount of missing data in the learn and test samples.
MISSING	Adds missing value indicators to the model. It has several forms. NO disables missing value indicators. YES will produce missing value indicators for all predictors in the model that have missing values in the learn sample. DISCRETE will produce missing value indicators only for discrete predictors. CONTINUOUS will do so only for continuous predictors. LIST=specifies a list of variables; those in the list that appear as predictors in the model and have missing values in the learn sample will get missing value indicators. LIST= can include variable groups and variables that are not part of the model.
VARDEF	Specifies whether a denominator of N or N-1 should be used in variance and standard deviation expressions in regression trees. The default is N, which is what the original CART implementation used.
PLC	Controls whether linear combinations other than the primary splitter are included in the node-by-node detail report (ignored unless the LCLIST command is in effect).

CVS Controls whether CV trees are saved in the GROVE.

Examples:

```
BOPTIONS SERULE=.85, SURROGATES=10, COPIOUS, LIST
BOPTIONS SPLITS=90, SURROGATES=8 PRINT=3, SERULE=0, OPTIONS
```

BUILD

Purpose

The **BUILD** command reads the data, chooses the LEARN and TEST samples (if any) and generates trees. It is the "hot" command that begins processing.

If using CART in the interactive mode (as opposed to a command file), the BUILD phase is ended with a QUIT command that returns you to CART.

The command syntax is:

```
BUILD
```

Examples:

```
USE SEATBELT.CSV  
MODEL BMW  
BUILD
```


CATEGORY

Purpose

The **CATEGORY** command indicates whether the target variable is categorical (thereby initiating a classification tree) and identifies which predictors are categorical.

The command syntax is:

```
CATEGORY <var1>, <var2>
```

Examples:

```
MODEL LOW
```

```
CATEGORY LOW (categorical dependent variable indicates CLASSIFICATION tree)
```

```
MODEL SEGMENT
```

```
CATEGORY SEGMENT
```

CATEGORY is also used to identify categorical predictor variables. CART will determine the number of distinct values for you. Example:

```
MODEL LOW
```

```
CATEGORY LOW, AGE, RACE, EDUC
```

CDF

Purpose

The **CDF** command evaluates one or more distribution, density, or inverse distribution functions at specified values.

For cumulative distribution functions the syntax is:

```
CDF [ NORMAL = z | T = t,dof | F = f,dof1,dof2 |
      CHI-SQUARE = chisq,dof | EXPONENTIAL = x | GAMMA = gamma,p |
      BETA = beta,p,q | LOGISTIC = x | STUDENTIZED = x,p,q |
      WEIBULL = x,p,q | BINOMIAL = x,p,q | POISSON = x,p ]
```

To generate density values, use the syntax above with the DENSITY option:

```
CDF DENSITY [ distribution_name = user-specified-value(s) ]
```

To generate inverse cdf values, specify an 'alpha' value between 0 and 1:

```
CDF INVERSE [ NORMAL=alpha | T=alpha,dof | POISSON=alpha,p |
      F=alpha,dof1,dof2 | CHI-SQUARE=alpha,dof | EXPONENTIAL=alpha |
      GAMMA=alpha,p | BETA=alpha,p,q | LOGISTIC=alpha |
      STUDENTIZED=alpha,p,q | WEIBULL = alpha,p,q |
      BINOMIAL=alpha,p,q ]
CDF NORMAL=-2.16, DENSITY NORMAL=-2.5, INVERSE CHISQ=.8,3
```

CHARSET

Purpose

The **CHARSET** command allows you to select which type of characters to use for character graphics (as opposed to high-resolution SYGRAPH graphics). You may choose either IBM screen and printer GRAPHICS characters or GENERIC characters that will print on any printer.

Caution: GRAPHICS characters do not print correctly on some printers; if you have problems, switch to GENERIC.

The command syntax is:

```
CHARSET GRAPHICS | GENERIC
```

Examples:

```
CHARSET GRAPHICS  
CHAR GENERIC
```

CLASS

Purpose

The **CLASS** command assigns labels to specific levels of categorical variables (target or predictor). Labels are not limited in their length, although in some reports they will be truncated due to space limitations. For instance, if variable DRINK takes on the values 0, 1, 2, and 3 in the data, you might wish to assign labels to those levels:

```
CATEGORY DRINK
CLASS DRINK 0=tea 1='Columbian coffee' 2="soda pop",
            3='Cold German Beer!'
```

Class labels will appear in the node detail, misclassification reports, terminal node reports, and in most instances where the numeric levels would normally show up, in lieu of the numeric levels themselves.

It is not necessary to specify labels for all levels of a categorical variable—any levels without a label will show up as numbers.

The command syntax is:

```
CLASS <variable> <level>=<string>, <level>=<string>, ...
```

You may issue separate CLASS commands for each variable, such as:

```
CLASS PARTY 1=Repub 2=Democratic 3="Peace and Freedom"
CLASS GENDER 0=female 1=male
CLASS EVAL$ "G"="Good", "F"="Fair", "P"="Poor"
```

or you may combine them in a single command, separating variables with a slash:

```
CLASS PARTY 1=Repub 2=Democratic,
            3="Peace and Freedom" / GENDER 0=female 1=male /,
EVAL$ "G"="Good", "F"="Fair", "P"="Poor"
```

Note that the label "Peace and Freedom" requires quotes, since it contains spaces. Labels consisting only of numbers and letters can be listed without quotes, but if so any letters will be converted to uppercase.

Note also that all class labels for a given variable must be defined at once, since the *<variable>* token that leads the list of classes clears out any existing class labels for the variable.

Variable groups that are composed of one type of variable only (i.e., numeric or character) may be used in the CLASS command similarly to variable names, e.g.:

```
GROUP CREDITEVAL = EVAL3MO, EVAL6MO, EVAL1YR, EVAL3YR
CATEGORY CREDITEVAL
CLASS CREDITEVAL 0="n/a", 1="Poor", 2="Fair", 3="Good"
```

Class labels are reset with the USE command. They are preserved in a CART grove file. They will not carry over from a BUILD run to a CASE run unless in a continuation of the BUILD session. To reset all class labels, issue the CLASS command with no options:

```
CLASS
```

To see a summary of class labels issue the command:

```
CLASS _TABLE_
```

COMBINE

Purpose

The **COMBINE** command begins a combined-tree or "committee of experts" run. All options for COMBINE are set with a previous instance of the MOPTIONS command.

The command syntax is:

```
COMBINE
```

Examples:

```
USE SEATBELT.CSV
MODEL BMW
MOPTIONS CYCLES = 10, EXPLORE = YES, DETAILS = NONE,
          RTABLES = NO, TRIES = 3, ARC = NO,
          SETASIDE = PROP = 0.100000
COMBINE
```

DATA

Purpose

The **DATA** command designates a block of statements to be interpreted as BASIC statements rather than as CART commands. The block is terminated with “DATA END.”

Example:

```
data
let mvq1=(mv<17)
let mvq2=(mv>=17 and mv<21.2)
let mvq3=(mv>=21.2 and mv<25)
let mvq4=(mv>=25)
let mvd=(mv>=21.2)
data end
```

DATAINFO

Purpose

The **DATAINFO** command generates descriptive statistics for numeric and character variables. Its simplest form is:

```
DATAINFO
```

The full command syntax is:

```
DATAINFO <varlist> / [ CHARACTER | NUMERIC,  
                      EXTREMES = <n>, TABLES ]
```

Examples:

To indicate particular variables:

```
DATAINFO GENDER$, WAGES, LOGWAGES
```

To generate statistics only for numeric variables, and for each such variable to list the extreme 15 values:

```
DATAINFO / NUMERIC, EXTREMES = 15
```

To produce full frequency tabulations, use the TABLES option:

```
DATAINFO POLPARTY$ / TABLES
```

To speed up the computation of statistics and avoid the (potentially time-consuming) complete tabulation of all variables, use the CONTINUOUS option to specify that only continuous statistics should be produced:

```
DATAINFO PROFIT, LOSS, VOLUME / CONTINUOUS
```

Variable groups may be used in the CATEGORY command similarly to variable names, e.g.:

```
GROUP GRADES = ROSHREC$, SOPHREC$, JUNIOR$, SENIOR$, PSAT, SAT, MCAT  
DATAINFO GRADES
```

Caution: if you have ordered variables (with many distinct values) included in the DATAINFO, the TABLES option can generate huge output.

The default is:

```
DATAINFO / EXTREMES = 5
```


DESCRIPTIVE

Purpose

The **DESCRIPTIVE** command specifies what statistics are computed and printed during the initial pass through the input data. The statistics will not appear in the output unless the command **LOPTIONS MEANS=YES** command is issued. By default, the mean, N, SD and sum of each variable will appear when **LOPTIONS MEANS=YES** is used. To indicate that only the N, MIN and MAX should appear in descriptive statistics tables, use the commands:

```
DESCRIPTIVE N, MIN, MAX  
LOPTIONS MEANS=YES
```

The command syntax is:

```
DESCRIPTIVE MEAN=<YES|NO>, N=<YES|NO>, SD=<YES|NO>, SUM=<YES|NO>,  
            MIN=<YES|NO>, MAX=<YES|NO>, MISSING=<YES|NO>, ALL
```

The ALL option will turn on all statistics and MISSING will produce the fraction of observations with missing data.

DISCRETE

Purpose

The **DISCRETE** command sets options specific to discrete or categorical variables.

The command syntax is:

```
DISCRETE [TABLES      = NONE      | SIMPLE | DETAILED ,
          CASE        = MIXED     | UPPER  | LOWER  ,
          MISSING     = MISSING   | LEGAL  ,
          REFERENCE   = FIRST     | LAST   ,
          MAX = <n,n> ,
          ORDER = YES|NO ,
          ALLLEVELS = YES|NO]
```

TABLES	Controls whether frequency tables should be printed following data preprocessing. SIMPLE generates a listing of the levels encountered for each discrete variable and total counts (across learn and test samples). DETAILED breaks down counts by learn and test sample, and also by the dependent variable for classification trees. The default is SIMPLE.
CASE	Controls whether character strings are case-converted. The default is MIXED.
MISSING	Controls whether missing values for discrete variables are treated as truly MISSING or are considered a legal and distinct level. LEGAL will process missing values for nontarget variables as legal. TARGET will process missing values for a model target only as legal. ALL will process missing values for all variables as legal.
REFERENCE	Specifies which level is considered the reference, or "left out" level. In MARS, a reference level is only needed when computing an OLS model for comparative purposes prior to the MARS model. By default, the FIRST level according to the ORDER and SORT criteria is considered the reference level. You may wish to change this to the LAST level to reach agreement with some other OLS programs.
MAX	Specifies the maximum number of distinct levels in discrete variables. The default is 20000,60000, which permits up to 20000 distinct classes for numeric variables and up to 60000 for character variables. You should only consider increasing this parameter if the program is unable to obtain a complete tabulation of one or more of your discrete variables.

ALLLEVELS By default, node statistics will not list discrete variable levels for a node that is not represented (N=0) in that node. Specifying ALLLEVELS=YES results in a complete tabulation of levels, including those with N=0 in the node.

ORDER Discrete variable splitters and cross validation for classification trees can be affected by the sorting of your dataset. ORDER=YES adjusts for any sorting in your data and should be used when comparing results between CART 5 or greater and previous versions of CART.

The default is

```
DISCRETE TABLES=SIMPLE, CASE=MIXED, MISSING=MISSING,  
REFERENCE=FIRST, ALLLEVELS=NO, ORDER=NO, MAX=20000,60000
```

DISALLOW

Purpose

The **DISALLOW** command specifies how predictor variables are constrained to be used, as primary splitters and/or as surrogates, at various depths of the tree and according to the node learn sample size. This command is only available in CART EX Pro (is ignored by other versions).

By default, all predictors are allowed to be used as primary splitters (i.e., competitors) and as surrogates at all depths and node sizes. For each predictor, the DISALLOW command is used to specify at which depths and in which partitions (by size) the predictor is NOT permitted to be used, either as a splitter, a surrogate, or both. The syntax is:

```
DISALLOW <variable> [ , <variable> , ... /
                        ABOVE = <depth> , BELOW = <depth> ,
                        MORE = <node_size> , FEWER = <node_size> ,
                        SPLIT | SURROGATE ]
```

To enable a DISALLOW command to apply to all variables, use the the syntax:

```
DISALLOW * [ / ABOVE = <depth> , BELOW = <depth> ,
                MORE = <node_size> , FEWER = <node_size> ,
                SPLIT | SURROGATE ]
```

Note that the ABOVE and BELOW options may be used together to describe the following depth ranges in which a variable is not used (D=depth):

ABOVE=N Variable will not be used if depth $D \leq N$, i.e., at depth N or shallower.

BELOW=M Variable will not be used if depth $D \geq M$, i.e., at depth M or deeper.

ABOVE=N, BELOW=M **N=>M**: This defines a depth range in which the variable will not be used, i.e., the variable will not be used if depth is between N and M, inclusive.

N<M: This defines two depth ranges in which the variable will not be used. The variable will not be used if $D \leq N$ (depth N and shallower) or if $D \geq M$ (depth M and deeper).

Similarly for the MORE and FEWER options, which operate on the node size (number of learn sample observations in the node being split, before any sub-sampling is done) rather than the depth:

MORE=N	Variable will not be used if the node has N or more records.
FEWER=M	Variable will not be used if the node has M or fewer records.

The DISALLOW command is cumulative. To reset all DISALLOW specifications (i.e., to return to the default), issue the empty command:

DISALLOW

Variable groups may be used in the DISALLOW command in the same manner as individual variable names.

Examples:

```
DISALLOW SEGMENT /ABOVE=3
DISALLOW REVM1 /ABOVE=1 SPLIT
DISALLOW CODES /ABOVE=3 SURROGATE
DISALLOW OHIGHT /BELOW=2
DISALLOW CODES /BELOW=2 ABOVE=3
DISALLOW CODES /FEWER=1000
```

ERROR

Purpose

The **ERROR** command specifies the method used to measure true regression error and misclassification rates.

The command syntax is:

```
ERROR [ CROSS = <n|var> | EXPLORATORY | PROPORTION = <x><y> |
      SEPVAR = <var> | FILE = <filename> ]
```

<x> is between 0 and 1, <n> is an integer, <var> is a variable and <filename> is any valid file.

CROSS	V-fold cross validation. You may indicate a number of CV cycles, in which case binning is carried out randomly while balancing on the target classes, or you may specify a variable for which each distinct value defines a CV bin.
EXPLORATORY	No independent testing—resubstitution estimate.
PROPORTION	Fraction of cases selected at random for testing, and optionally, validation.
SEPVAR	Named variable separates learn, test, and validation samples. The test value is 1 for numeric SEPVAR variables and "TEST" or "test" for character SEPVAR variables. For the validation sample the values are -1 (numeric) and "VALID," "Valid" or "valid."
FILE	Test sample is contained in a separate data file. For details on naming conventions, see the reference for the USE command.

Examples:

```
ERROR CROSS=10 (the default method for CART models)
ERROR PROPORTION=.25 (select 25% of cases at random for test)
ERROR FILE=SHARP (test cases are found in file SHARP.SYS)
ERROR PROPORTION=.3, .2 (30% testing, 20% validation/scoring)
ERROR CROSS=MYBINS (the variable MYBINS contains the CV fold assignments)
```

EXCLUDE

Purpose

The **EXCLUDE** command specifies a list of independent variables to exclude from the analysis. In other words, all variables other than the target and those listed in EXCLUDE and WEIGHT commands will be used as predictors.

The command syntax is:

```
EXCLUDE <varlist>
```

in which <varlist> is a list of variables NOT to be used in the model-building process. All other variables will be used.

See the MODEL and KEEP commands for other ways to restrict the list of candidate predictor variables.

Examples:

```
MODEL CHOICE
```

```
EXCLUDE ID, SSN, ATTITUDE (all numeric variables except ID, SSN and  
ATTITUDE can be used in the CART process)
```

FORCE

Purpose

FORCE identifies CART splits to be implemented at the root and first child nodes, in lieu of the splits that CART would naturally determine based on the learn data. The FORCE command applies to CART trees only. Its syntax is:

```
FORCE ROOT|LEFT|RIGHT ON <predictor> AT <splits>
```

For example:

```
FORCE ROOT ON GENDER$ AT "Male", "Unknown"  
FORCE LEFT ON REGION AT 0,3,4,7,999  
FORCE RIGHT ON INCOME AT 100000
```

To reset forced splits, use the command with no options

```
FORCE
```


FPATH

Purpose

The **FPATH** command sets the default search path for unquoted file names. Its syntax is:

```
FPATH "<file prefix or path>" [/OUTPUT SAVE SUBMIT GROVE USE]
```

OUTPUT	Set the default path for classic text output files specified with the OUTPUT command.
SAVE	Set the default path for output datasets specified with the SAVE command.
SUBMIT	Set the default path for command files to be executed via the SUBMIT command.
GROVE	Set the default path for grove files (either input or output).
USE	Set the default path for input datasets specified with the USE or ERROR FILE commands.

If no options are specified, the path indicated applies to all file types. If no path is given, the existing path is replaced by the default, which is the current working directory. The FPATH command has no effect on quoted file names.

FORMAT

Purpose

The **FORMAT** command controls the number of digits that are displayed to the right of the decimal point in analysis output. You may select from 1 to 9 digits, or 0 digits, or -1 for no digits and no decimal point. The default is 3.

The UNDERFLOW option prints tiny numbers (those that would appear to be zero in the chosen precision) in scientific (exponential) notation.

The command syntax is:

FORMAT <#> [/UNDERFLOW]

Examples:

FORMAT=5

FORMAT=0

FORMAT=9/UNDERFLOW (print tiny numbers with exponents)

GROUP

Purpose

The **GROUP** command defines variable groups.

The command syntax is:

```
GROUP <groupname> = <variable> <variable> ...
```

Group names are used like variable names in commands that process variable lists, resulting in more compact lists. The following commands set up three groups and use them in the **KEEP**, **CATEGORY**, and **CLASS** commands (along with variables **SEGMENT**, **AGE**, **PROFIT**) for a three-level classification tree model:

```
GROUP DEMOGRAPHICS = GENDER RACE$ REGION$ PARTY EDUCLEV
GROUP CREDITINFO   = FICO1 FICO2 TRW LOANAMOUNT AUTOPAYMENT,
                    MORTGAGEAMOUNT MORTGAGEPAY
GROUP CREDITRANK    = RANKVER1 RANKVER2 RANKVER3
CATEGORY DEMOGRAPHICS TARGET$ SEGMENT CREDITRANK
CLASS CREDITRANK 0="Not available", 1="Poor", 2="Good",
                 3="Excellent"
MODEL TARGET$
KEEP DEMOGRAPHICS CREDITINFO SEGMENT CREDITRANK
MART GO
```

Groups can contain a mix of character and numeric variables; however, the **CLASS** command will accept homogenous (all character or all numeric) groups only. A variable may be included in more than one group. If a group is assigned a name that is identical to a variable name, the group name will take precedence in variable lists (i.e., the variable name will be masked).

The following commands recognize variable groups:

```
CATEGORY, KEEP, EXCLUDE, AUXILIARY, IDVAR, CONSTRAIN
DATAINFO, PENALTY, CLASS, XYPLOT, HISTOGRAM
```

GROVE

Purpose

The **GROVE** command names a grove file in which to store the next tree (or committee or group of impute trees) or to use in the next TRANSLATE or SCORE operation. If an unquoted name is given without an extension, ".GRV" is appended.

The command syntax is:

```
GROVE <filename> [IMPORT="legacy treefile" LOAD MEMO="contents" ECHO]
```

Examples:

```
GROVE "c:\modeling\rev1\groves\M_2b.grv"  
GROVE MOD1
```

To convert a legacy "treefile" (e.g., mytree.tr1) from a previous version of CART to a grove, use the IMPORT option, e.g.:

```
GROVE "\robustus\projects\groves\J3b.grv" IMPORT="c:\c3po\legacy.tr1"
```

To test a grove file for validity, use the LOAD option, e.g.:

```
GROVE "qmodel1.grv" LOAD
```

If the grove file is invalid, an error message will be generated.

To add a memo to a grove command, use the MEMO option, e.g.:

```
GROVE "filename.grv" MEMO="A one-line quoted memo"
```

To view any memo that may be embedded in a particular grove, use the ECHO option, e.g.:

```
GROVE "filename.grv" ECHO
```

If one of the above options is specified, the file name must be quoted.

HARVEST

Purpose

The **HARVEST** command specifies which trees in a grove are processed (during SCORE or TRANSLATE) and how those trees are pruned for processing. For selecting trees in a grove, the HARVEST SELECT command is used.

The command syntax is:

```
HARVEST SELECT [ ALL | RELERR = <x> | COMPLEXITY = <x> |
                NODES = <n> | RANDOM = <n> |
                KEEP = <n1,n2,...> | EXCLUDE = <n1,n2,...> ,
                BEST = <n> ]
```

If the HARVEST SELECT command is not issued, all trees in the grove are selected.

HARVEST SELECT is used to select specific trees from multi-tree models created with the COMBINE command, or from groves containing batteries of trees requested with the BATTERY command. Since regular CART models have only a single tree, HARVEST SELECT has no effect on them (use HARVEST PRUNE instead).

Prior to being used in a scoring or translation step, the selected trees are pruned to their optimal size. To specify a pruning condition to be applied to all the selected trees, use the HARVEST PRUNE command.

The command syntax is:

```
HARVEST PRUNE [ NODES = <n> | DEPTH = <n> |
                TREENUMBER = <n> | COMPLEXITY = <x> ]
```

If several trees are selected, you may list different pruning criteria for each with the HARVEST PRUNE LIST command

The command syntax is:

```
HARVEST PRUNE LIST [ NODES = <n1,n2,...> | DEPTH = <n1,n2,...> |
                    TREENUMBER = <n1,n2,...> ]
```

The options on the HARVEST SELECT command are:

ALL	Select all trees in the grove.
RELERR=<x>	Select all trees which, when pruned to optimal size, have a test sample relative error rate (or resubstitution error rate if no test sample was used) less than <x>.
COMPLEXITY=<x>	Select all trees which, when pruned to optimal size, have a complexity threshold less than <x>.
NODES=<n>	Select all trees which, when pruned to optimal size, have less than or equal to <n> terminal nodes.
RANDOM=<n>	Randomly select up to <n> trees from the grove.
DEPTH=<n>	Select all trees which, when pruned to optimal size, are less than or equal to <n> nodes deep.
BEST=<n>	When used with the RELERR, COMPLEXITY, NODES, RANDOM, KEEP, or EXCLUDE criterion, ensures that only the most accurate <n> trees are selected from those meeting the original criterion. Accuracy is based on test sample error rate (or resubstitution error rate if no test sample was used).

HARVEST CVTREES=YES|NO specifies whether ancillary trees created as part of a CART cross-validation model are selected. By default, they are not.

A new grovefile, containing only the harvested trees, may be created with the OUTPUT option, for example:

```
HARVEST SELECT KEEP=5 OUTPUT="justone.grv"
```

Examples:

```
USE "gymtutor.csv"
SAVE "testPRED.CSV" / MODEL
GROVE "BUILD_GYMc.GRV"
HARVEST PRUNE TREENUMBER = 1
SCORE
```

HELP

Purpose

The **HELP** command provides information about CART commands. You can abbreviate the name of the command.

The command syntax is:

HELP [*<command>*]

Examples:

HELP (lists commands available for the current procedure)

HELP HELP (provides information on the HELP command)

HISTOGRAM

Purpose

The **HISTOGRAM** command produces low resolution density plots.

The command syntax is:

```
HISTOGRAM <var1> [, <var2> , <var3> , ... ,  
                / FULL, TICKS | GRID, WEIGHTED, NORMALIZED, BIG ]
```

The plot is normally a half screen high: the FULL and BIG options will increase it to a full screen (24 lines) or a full page (60 lines).

TICKS and GRID add two kinds of horizontal and vertical grids.

WEIGHTED requests plots weighted by the WEIGHT command variable.

NORMALIZED scales the vertical axis to 0 to 1 (or -1 to 1).

Examples:

```
HISTOGRAM IQ / FULL, GRID  
HISTOGRAM LEVEL(4-7) / NORMALIZED
```

Only numerical variables may be specified.

Variable groups may be used in the HISTOGRAM command similarly to variable names.

IDVAR

Purpose

The **IDVAR** command lists extra variables to save in the next dataset to be SAVED. These can be any variables from the USE dataset that are not in the model. (Model variables are saved with the SAVE / MODEL option.)

The command syntax is:

If every case in your file has a unique identifier, say SSN, you could specify:

```
IDVAR SSN  
SAVE "WATER.CSV"
```

The file WATER.CSV will include the variable SSN in addition to its normal contents.

If you want to include all the non-model and model variables in the saved dataset, you would issue:

```
IDVAR / ALL  
SAVE <"filename"> / MODEL
```

Variable groups may be used in the IDVAR command similarly to variable names.

KEEP

Purpose

The **KEEP** command specifies a list of independent variables.

The command syntax is:

```
KEEP <indep_list>
```

in which <indep_list> is a list of potential predictor variables. If no <indep_list> is specified, all numeric variables are considered for node splitting (unless an EXCLUDE command or <indep_list> is included on the MODEL statement).

Independent variables may be separated by spaces, commas, or + signs. A range of variables may be specified with the first and last variables (in data set order) separated by a dash.

See the MODEL and EXCLUDE commands for other ways to restrict the list of candidate predictor variables.

Examples:

```
MODEL CLASS
```

```
KEEP AGE-IQ, EDUC, FACTOR(3-8), RACE (selected variables)
```

```
MODEL CHOICE
```

```
KEEP FOOD+AGE+HEIGHT-WAIST
```

LABEL

Purpose

The **LABEL** command defines variable labels. Labels are not limited in length, although in some reports they will be truncated due to space limitations.

The command syntax is:

```
TABLE <variable>="ADD LABEL IN QUOTES"
```

Examples:

```
LABEL RESPONSE="Did subject purchase at least one item? 1=yes, 0=no"
```

or

```
LABEL PARTY$="Political affiliation, sourced from public database."
```

If labels are imbedded in your dataset (such as SAS(tm) datasets), they will be used in CART and there is no need for you to issue LABEL commands unless you wish to change or remove them.

Variable groups may be used in the LABEL command similarly to variable names.

To see a summary of variable labels, issue the command:

```
LABEL _TABLE_
```

LCLIST

Purpose

The **LCLIST** command identifies a group of continuous predictors among which CART should attempt to produce a linear combination at each node. The **LINEAR** command is now deprecated in favor of **LCLIST**. Its syntax is:

```
LCLIST <varlist> [ / <options> ]
```

in which <varlist> can be an explicit list of continuous predictors or the **_KEEP_** keyword (shorthand for whatever the keep list is for the model). Some examples:

```
LCLIST credit_score,rate,rebate
LCLIST _keep_
LCLIST x,y,z / N=100, EXH=YES
```

To clear out all LCLISTs, simply issue the **LCLIST** command alone:

```
LCLIST
```

Multiple **LCLIST** commands can be issued. In this way, multiple linear combinations may be developed at each node. The linear combination with the highest improvement will be compared to the best univariate splitter to determine the primary splitter in the node. Options are:

- | | |
|-------------------------|---|
| N=<n> | Specifies the minimum number of records required in a node for linear combination splits from this LCLIST to be considered. Smaller nodes will not consider this LCLIST. This is essentially an LCLIST-specific atom. Default=3. |
| W=<x> | Similar to N=<n> but based on sum of case weights. If this option is issued, a node must have a sum of case weights equal to or exceeding <x> for this LCLIST to be considered. This is essentially an LCLIST-specific weighted atom. |
| SIZE=<n> | The maximum number of predictors in a linear combination. Must be > 1. The default is 6. |
| STORED=<n> | Defines how many candidate linear combinations formed from the LCLIST are maintained in memory during the search. A high value allows for a more comprehensive search involving higher-ordered linear combinations, but at a potentially significant increase in compute time. Must be > 1. The default is 5. |
| OPTIM=<n> | Must be 0 or greater. The default is 0. |

- PENALTY=<x>** Must be in the range [0.5, 1.0] inclusive. Defaults to 0.9.
- POSITIVE=<yes|no>** Specifies whether all coefficients must be constrained to be positive. The default is NO.
- DELETE=<x>** Governs the backwards deletion of variables in a the stepwise linear combination search algorithm. The default is 0.20.
- DOF=<x>** When comparing a linear combination against univariate competitors, the LC improvement is DOF-adjusted:
- $$\text{adj_imp} = \text{improvement} * (N - X * (NC - 1) - 2) / (N - 2)$$
- in which:
- N** = number of records used in the LC search algorithm (usually the node size)
- NC** = number of nonzero coefficients in the LC
- improvement** = unadjusted improvement, displayed in model results, reports, etc.
- X** = parameter specified on the DOF option.
- For agreement with previous versions of CART (that used the the LINEAR command), use DOF=1. To disable the adjustment, use DOF=0. The default is 1.0.
- EXH=<yes|no>** Tells CART to repeat the stepwise search algorithm using each predictor in the LCLIST as the focal variable. This increases compute time proportional to the number of predictors in the LCLIST. It can, in some cases, yield better split points than the default approach. Default=NO.
- SS=<yes|no>** The default (SS=yes) allows the linear combination search algorithm to proceed even if some of the predictors in the LCLIST have a high proportion of missing values or are constant. Disabling this feature (SS=no) causes CART to use a more stringent, listwise-like criterion for determine which records in a node are used in forming linear combinations and whether linear combination searching is even attempted in a node for this LCLIST.
- SEARCH=<n>** Limits the linear combination search to only consider the topmost N univariate competitors in the LCLIST. The default is 10, the minimum value is 2. Smaller values reduce run time at

the expense of perhaps not considering potentially valuable linear combinations.

Examples:

LCLIST _KEEP_ (Enable LCs; allow all predictors to be considered)

LCLIST CRIM ZN INDUS CHAS /N=50 (Specify LCLIST; Set min. node size to 50)

LIMIT

Purpose

The **LIMIT** command allows tree growth limits to be set.

The command syntax is:

```
LIMIT ATOM=<n>, SUBSAMPLE=<n>, NODES=<n|AUTO>,
      DEPTH=<n|AUTO>, LEARN=<n|AUTO>, TEST=<n|AUTO>,
      DATASET=<n>, ERRORSET=<n>,
      MINCHILD=<n>
```

in which <n> is a whole number.

ATOM	Minimum size below which a node will not be split. Default=10.
SUBSAMPLE	Node size above which a subsample is used to locate splits.
NODES	Forecast of the number of terminal nodes in the largest tree grown. Default of AUTO lets CART set a value for you. Override allocates required workspace for unusual problems.
DEPTH	Limits maximal tree growth to a specified depth. Default of AUTO forecasts depth of largest tree likely to be grown.
LEARN	Maximum number of cases to allow into the learning set. By default, no limit is in effect. AUTO removes current limit.
TEST	Maximum number of cases to allow into the test set. By default, no limit is in effect. AUTO removes current limit.
MINCHILD	Sets the minimum size for a child node. The default is 1.
WMINCHILD	Sets the minimum weighted size for a child node. It is only used if you explicitly set a nonzero value.

Examples:

```
LIMIT LEARN=20000, TEST=5000
LIMIT ATOM=15, NODES=150, LIST
LIMIT DEPTH=18, MINCHILD=10, WMINCHILD=30
```

On some platforms, CART can automatically determine the number of records in the USE= and ERROR FILE= datasets, but on other platforms it cannot and will assume 1000 records. These assumptions may lead to poor choices of memory parameters if your datasets have considerably more records than 1000. In this case, use the DATASET and ERRORSET options to inform CART of the correct number of records in your datasets. Some examples are:

```
LIMIT DATASET=33000  
LIMIT DATASET=100000, ERRORSET=75000
```


LINEAR

Purpose

The **LINEAR** command allows CART to search for linear combinations of non-categorical predictor variables to split nodes.

The command syntax is:

```
LINEAR N=<n1>, DELETE=<x>, LINSPLITS=<n2|AUTO>, EXHAUSTIVE
```

in which <x> is a fractional or whole number and <n1> and <n2> are whole numbers.

- | | |
|-------------------|---|
| N | specifies the minimum number of cases required in a node for linear combination splits to be considered. Smaller nodes will be split on single variables. |
| DELETE | governs the backwards deletion of variables in a stepwise algorithm. The default is 0.20. |
| LINSPLITS | is a forecast of the maximum number of linear combination splits in the maximal tree. This value is estimated automatically by CART and normally need not be set. The automatic estimate may be overridden to allocate more linear combination workspace. |
| EXHAUSTIVE | tells CART to attempt computing linear combinations using each continuous independent variable as the "perturbation" variable. |

Examples:

```
LINEAR N=400, DELETE=.30
```

Linear combination splits are turned off by simply entering the command

```
LINEAR
```

The **LINEAR** command is deprecated in favor of the **LCLIST** command, and may be removed from future versions of CART.

LOPTIONS

Purpose

The **LOPTIONS** command toggles several “logical” options on and off.

The command syntax is:

```
LOPTIONS MEANS=YES|NO, TIMING=YES|NO, NOPRINT
          PREDICTION_SUCCESS=YES|NO,
          GAINS=YES|NO, ROC=YES|NO, PS=YES|NO,
          PLOTS = YES|NO / "<plot_character>",
          DBMSCOPY = YES|NO, STATTRAN = YES|NO
```

MEANS	Controls printing of summary stats for all model variables.
TIMING	Reports CPU time on selected platforms.
NOPRINT	Omits node-specific output and prints only summary tables.
PREDICTIONS	Requests the prediction success table.
GAINS	Toggles the printing of gains charts in CART for classification models. Binary models always show these charts.
ROC	Toggles the printing of ROC charts in CART for classification models. Binary models always show these charts.
PS	Toggles printing of the pruning sequence when a tree is built.
PLOTS	Toggles summary plots and allows a user-specified plotting symbol.
DBMSCOPY	Toggles support for the DBMS/COPY data access engine (deprecated).
STATTRAN	Toggles support for the Stat/Transfer data access engine.

To turn an option ON the '=YES' portion is not needed.

Examples:

```
LOPTIONS MEANS          (turn MEANS printing on)
LOPTIONS MEANS=NO      (turn MEANS printing off)
```

MEMO

Purpose

The MEMO command defines a text memo that is saved with the model. A memo is cumulative until an analysis is performed, after which the memo is reset. Enclosing the content of a memo in quotes is not necessary; however, case is preserved and certain punctuation marks (e.g., apostrophes) are better handled if the text is quoted.

Examples:

A two-line memo in which the first line has case preserved (by using quotes) and the second does not:

```
MEMO "This is my memo, line one."
MEMO a second line, will display entirely in uppercase.
```

A memo composed of a group of lines ending with the END tag, which will add three lines to any existing memo:

```
MEMO
"This model focuses on IRR and income variables in Sept '03."
"A series of variable shaving models result, CART and TreeNet"
"engines, analysis data version 2a."
END
```

To see the currently-defined memo, issue the command

```
MEMO ECHO
```

To reset the memo:

```
MEMO RESET
```

Normally, memos are reset after a model is built. To force the memo to persist across models until it is explicitly RESET, use the command

```
MEMO PERSIST=YES (PERSIST=NO returns to the default)
```

To cause the memo to be displayed in the classic text output at the start of each model, use the INCLUDE option:

```
MEMO INCLUDE=YES (INCLUDE=NO returns to the default)
```

To quickly see any memo that may be embedded in a particular grove, use the ECHO command on the GROVE command:

```
GROVE "filename.grv" ECHO
```

As an alternative to the MEMO command, you can specify a single line, quoted memo on the GROVE command itself:

```
GROVE "filename.grv" MEMO="A one-line quoted memo"
```

MEMORY

Purpose

The **MEMORY** command provides information about memory usage and memory requirements for the current model. Use the **BOPTIONS**, **LIMIT** and **ADJUST** commands to refine your problem to fit it into available memory.

The command syntax is:

MEMORY

METHOD

Purpose

The **METHOD** command specifies the splitting rule used in tree construction.

The CLASSIFICATION tree command syntax is:

```
METHOD [ GINI | SYMGINI | TWOING | ORDERED | PROB | ENTROPY,
POWER=<x> ]
```

GINI Is the default and is frequently the best choice.

SYMGINI May be used with variable misclassification costs.

TWOING Is a competitor to GINI.

ORDERED Can be used for ordered categorical dependent variables.

PROB Requests probability trees instead of classification trees.

ENTROPY Is a modification of GINI, using $p \cdot \log(p)$ rather than $p \cdot (1-p)$.

POWER=<x> Can be used to tune CART away from end-cut splits.

The REGRESSION tree command syntax is:

```
METHOD [ LS | LAD ]
```

LS uses a least squares measure of within-node dispersion and

LAD uses a least absolute deviation measure.

Examples:

```
METHOD TWOING (use TWOING for classification)
```

```
METHOD LAD (use LAD for regression)
```

```
METHOD ENTROPY,LS (use ENTROPY for classification and least squares for
regression)
```

MISCLASS

Purpose

The **MISCLASS** command specifies misclassification costs.

The command syntax is:

To specify unit misclassification costs, use one of the following commands.

```
MISCLASS = UNIT
```

To specify other than unit costs, use one of the following command forms

```
MISCLASS COST = <x> CLASSIFY <n1,n2,...> AS <m> [ / COST = ..  
CLASSIFY .]  
MISCLASS COST = <x> CLASSIFY <n> AS <m1,m2,...> [ / COST = ..  
CLASSIFY .]
```

in which *<depvar>* is the dependent variable and *<indep_list>* is an optional list of potential predictor variables. If no *<indep_list>* is specified, all variables are used for CART processing (unless KEEP or EXCLUDE commands are used).

Examples:

The cost of misclassifying a class 2 case as a class 4 case is 4.5:

```
MISCLASS COST=4.5 CLASSIFY 2 AS 4
```

The cost of misclassifying a case from classes 1, 2, 3, 5 or 8 as a class 6 case is 2.75:

```
MISCLASS COST=2.75 CLASSIFY 1-3,5,8 AS 6
```

MISCLASS commands are cumulative—each command will specify a part of the misclassification matrix. To reset the matrix use:

```
MISCLASS UNIT
```

MODEL

Purpose

The **MODEL** command specifies the dependent variable.

The command syntax is:

```
MODEL <depvar> [ = <indep_list> ]
```

in which <depvar> is the dependent variable and <indep_list> is an optional list of potential predictor variables. If no <indep_list> is specified, all variables are used for CART processing (unless KEEP or EXCLUDE commands are used).

Examples:

```
MODEL DIGIT      (all non-character variables used in tree generation)
MODEL WAGE = AGE - IQ , EDUC, FACTOR(3-8) , RACE      (selected variables)
MODEL CLASS = PRED(8) + VARA-VARZ + PRED(1-3)
```

See the KEEP and EXCLUDE commands for another way to restrict the list of candidate predictor variables.

MOPTIONS

Purpose

The **MOPTIONS** command sets options for a subsequent **COMBINE** command (which launches the building of combined or multi-trees, a committee of experts tree). The data are split into a "setaside" set and an "overall" set. Trees are built and pruned using "overall" data, and are evaluated using "setaside" data. Learn and test samples for each of the trees in the expert series are constructed from the "overall" set.

These samples may be copies of the "overall" data, or may be sampled with or without replacement from the "overall" set. It is not necessary to have a test set for each tree—they can be built using cross-validation or with no pruning (exploratory). It is not necessary to have a "setaside" set, although without it comparison of the initial tree and the expert set must be done with two additional, separate case runs.

The command syntax is:

```
MOPTIONS CYCLES=<N>,  ARC=<yes|no>,
          SETASIDE=PROP=<x> | FILE=<file> | SEPVAR=<var>,
          TEST | CROSS=<N> | EXPLORE,
          DETAILS=INITIAL|SET|ALL|NONE,
          TRIES=<N>,  POWER=<X>,
          RTABLES=<yes|no>
```

- | | |
|---------------------------|--|
| CYCLES | specifies the number of desired trees in the committee of experts, not including any initial tree. |
| ARC | specifies which combine method will be used. When ARC=YES, the ARCing (Adaptive Resampling and Combining) method is used. When ARC=NO, the bootstrap aggregation (or bagging) method is used. Default is ARC=NO. |
| SETASIDE | specifies how the "setaside" sample is created. This sample is NOT used to build or prune any of the trees. It is used to evaluate the predictive capability of trees only, including the initial tree. |
| PROP=<x> | specifies the proportion (0 to 1) drawn from the USE data. |
| FILE=<file> | sets up a separate dataset. |
| SEPVAR=<var> | separates the learn and test samples with a named variable. The "setaside" value is 1 for numeric and "SETASIDE" or "setaside" for character variables. |

The TEST, CROSS and EXPLORE options are used to specify if, and how, pruning is conducted. They are mutually exclusive options.

TEST	specifies that the unsampled training data is to be used as a test sample to prune each tree.
CROSS	specifies that N-fold cross validation is used for each tree in the series, in lieu of a test sample. If <N> is not specified, it defaults to 10.
EXPLORE	specifies that no test sample or cross validation is to be used for each tree.
TRIES	Occasionally, CART cannot build one of the trees in the series. You can specify how many times CART should draw and redraw learn and test samples in an effort to get it built. The default is 3.
POWER	This is the exponent K in the ARC function, evaluated for each observation in the overall set: $arc_func = \frac{(1 + m(i)^k)}{\sum_j (1 + m(j)^k)}$ A value of 0 effectively turns ARC off.
RTABLES	Controls the tables CART can produce to summarize how observations in the overall set are being repeated into the learn and test samples, both for each tree and cumulatively at the end of the series.
DETAILS	controls whether CART produces detailed output (tree sequence, node details, etc.) for the initial tree and for each tree in the series.

Examples:

```
MOPTIONS CYCLES = 10, EXPLORE = YES, DETAILS = NONE, RTABLES = NO,
TRIES = 3, ARC = NO, SETASIDE = FILE = "C:\gymtutorTEST.csv"
```

NAMES

Purpose

The **NAMES** command lists the variables on the data set.

The command syntax is:

NAMES

NEW

Purpose

The **NEW** command resets all CART-specific options while leaving CART's global options (USE file, PRINT settings, etc.) in effect.

The command syntax is:

NEW

NOTE

Purpose

The **NOTE** command lets you write comments on your output. A note can span any number of lines, but no line may be more than 150-characters long. You can embed an apostrophe in a note if you enclose the line in double quotation marks. You can embed double quotation marks if you enclose the line in apostrophes (single quotation marks). A number without quotation marks sends the corresponding ASCII character to the current output device.

The command syntax is:

```
NOTE <#> '<$>', '<...>', <#>
```

Examples:

```
NOTE 'THIS IS A COMMENT.' 'This is second line of comment.',  
    "It's the third line here!"
```

```
NOTE 'This the top of a new page' (subsequent NOTE creates line break).
```

OPTIONS

Purpose

The **OPTIONS** command displays the CART options currently in effect, including the currently-used file, any weighting, grouping or selection in effect, short, medium or long output, current graphics character set, number of decimal places to which output prints, and the output destination.

The command syntax is:

OPTIONS

OUTPUT

Purpose

The **OUTPUT** command routes output to the screen (the video display) or to a file. If you send output to a file and specify a simple filename, CART automatically gives the file a ".DAT" extension. If you supply a complete path name for the file you must enclose the name in quotes. If you send output to a file, the analysis results will also appear on the display.

If the screen pauses waiting for you to hit [Enter] or [Return], output to a file will also pause.

The command syntax is:

```
OUTPUT * | <file>
```

Examples:

```
OUTPUT *           (sends subsequent output to screen only)
OUTPUT FILE1       (sends output to FILE1.DAT in the default directory)
OUTPUT 'C:\REPORTS\NEWOUT.DAT'
```

PARTITION

Purpose

The **PARTITION** command defines how a single input dataset is to be partitioned into learn, test and validation samples. There are two options: specify the proportions numerically or specify a variable that identifies the sample into which each record should be placed.

```
PARTITION [ LEARN = <x>, TEST = <x>, VALIDATION = <x> ]  
PARTITION SEPVAR = <variable>
```

For instance, to specify that 20% of the data should be allocated for testing purposes and 25% as validation data:

```
PARTITION TEST=.2, VALID=.25
```

In the above example, the LEARN option does not appear so the amounts specified for test and validation samples must be expressed as proportions between 0 and 1 and must sum to less than 1. If you specify the LEARN option, then the amounts will be normalized to sum to 1.0, such as in:

```
PARTITION LEARN=20, TEST=12, VALID=8
```

Which would result in 50% of the data for the learn sample, 30% for the test sample and 20% for the validation sample.

```
PARTITION SEPVAR=PURPOSE$
```

specifies a character variable that should take on values "TEST," "Test," "VALID" or "Valid" to steer records into the test and validation samples, otherwise they will go to the learn sample. For a numeric separation variable, such as

```
PARTITION SEPVAR=USAGE
```

a value of 1 will place the record into the test sample and -1 for the validation sample.

PENALTY

Purpose

The **PENALTY** command offers three ways to specify a multiplicative fraction between 0 and 1 to "penalize" (down-weight) the improvement, thus making it more difficult for the variable to be chosen as the primary splitter in relation to other predictor variables.

Predictor-specific improvement factor

By default, no variable-specific penalty is applied to a variable's improvement when considering the variable as a splitter (although a penalty for missing data may be in effect).

The command syntax is:

```
PENALTY <var1> = <pen1>, <var2> = <pen2>, ...
```

in which the improvement evaluated for <var1> is multiplied by 1-<pen1>.

Two additional types of improvement penalties may be specified. The MISSING and HCC options may be given after the slash.

The command syntax is:

```
PENALTY <var> = <pen> ... / MISSING = <xm1>, <xm2>, HCC = <xh1>, <xh2>
```

Missing value improvement penalty

To penalize variables that have a large proportion of missing values in the partition (node) being split, the MISSING option is used. This option allows significance of the primary splitters and all competitors to be weighted by a simple function of the percentage of cases present (nonmissing) in the node partition. The expression for weighting the significance is:

$$improvement = improvement * factor$$

in which *factor*=1.0 if there are no missing values and

$$factor = xm1 * (fract ^ xm2)$$

if there are missing values. *Fract* is the proportion of observations in the partition (node) that have nonmissing values for the splitter in question. If *xm1* and *xm2* are

set to values that result in taking a root of a negative number, or result in $improvement < 0$, $improvement$ is set to 0. If $improvement > 1$, it is set to 1.

High-Level Categorical Improvement Penalty

To penalize categorical splitters that have a high number of levels relative to the number of records in the partition (node), the HCC option is used. Consider the expression:

$$ratio = \log_base_2 (N \text{ records in node}) / (N \text{ categories} - 1)$$

The HCC option weights the improvement of primary splitters and all competitors by the following function:

$$improvement = improvement * factor$$

in which $factor=1.0$ if $ratio \Rightarrow 1.0$ and

$$factor = 1 - xh1 + xh1 * (ratio ^{xh2})$$

if $ratio < 1.0$. If $xh1$ and $xh2$ are set to values that result in taking a root of a negative number, or result in $improvement < 0$, $improvement$ is set to 0. If $improvement > 1$, it is set to 1.

By default, improvement penalties are applied to surrogates in the same way that they are applied to competitors. To disable penalties for surrogates, use the command:

```
PENALTY / SURROGATE=NO
```

Variable groups may be used in the PENALTY command similarly to variable names.

The default values are:

```
MISSING=1.0,0.0 , HCC=1.0,0.0 , SURROGATE=YES.
```

Examples:

```
PENALTY NFAMMEM = .75, TANNING = .25 /,
      MISSING = 0.50, 0.75,
      HLC = 1.00, 3.75
```

PRIORS

Purpose

The **PRIORS** command specifies prior class probabilities for classification trees.

The command syntax is:

```
PRIORS [ DATA | LEARN | TEST | EQUAL | MIX |
        SPECIFY <class1>=<x1>, <class2>=<x2>, ... ]
```

in which <x1>, <x2>, ... is a vector of real numbers. The options set prior class probabilities as follows:

DATA	priors match observed sample shares in combined learn and test data.
LEARN	priors match observed sample shares in learn data alone.
TEST	priors match observed sample shares in test data alone.
EQUAL	uniform priors, automatically set to 1 / (number of classes).
MIX	priors set to the average of DATA and EQUAL options.
SPECIFY	<class1>=<x1>,<class2>=<x2>,... priors set to any strictly positive numbers. CART will normalize the values to sum to 1.0. A value must be assigned to each class. For character classes, the class value must be in quotes. The SPECIFY option requires that the dependent variable already be identified on the MODEL command.

Examples:

```
PRIORS SPECIFY "COKE"=1, "Pepsi"=2,
              "H2O"=4, "7UP"=1      (explicit list, let CART rescale)
PRIORS EQUAL                                (the default)
PRIORS MIX                                (split the difference between DATA and EQUAL)
```

PRINT

Purpose

The **PRINT** command switches you between standard and extended analysis results for certain procedures.

The command syntax is:

```
PRINT SHORT | LONG | MEDIUM
```

Examples:

```
PRINT=SHORT (Produces only standard output from commands)  
PRINT=LONG (Prints extended output for some procedures)
```

QUIT

Purpose

The **QUIT** command ends your CART session.

The command syntax is:

QUIT

The QUIT command will terminate the GUI, so you probably do not want it at the end of command files intended to be run there via the “Submit Window” facility. Console versions of CART running in batch mode will terminate automatically once all commands have been processed. Any commands appearing in a command file after a QUIT command will be ignored.

REM

Purpose

The **REM** command is for comments. All subsequent text on that line is ignored. The REM command is especially useful when writing programs in BASIC and in the writing of command files.

The command syntax is:

```
REM <text>
```

Examples:

```
REM This is a comment line and is not executed
```

RUN

Purpose

RUN processes the input dataset(s), produces summary reports, and optionally creates two output datasets, but no modeling is done. Its syntax is:

```
RUN [SD = "saved_dataset" PD = "processed_dataset" PDM = <yes|no>]
```

The PDM option governs whether internal class labels are written to the preprocessed dataset (PDM=YES), rather than the original ones (PDM=NO, which is the default). The saved dataset can alternately be specified with the SAVE command.

Examples:

```
REM Create a new dataset from the old one by adding a new variable
REM and deleting some records
USE INFILE.CSV
SAVE OUTFILE.CSV
%IF DEATHDATE=. OR BIRTHDATE=. THEN DELETE
%LET DEATHAGE=(DEATHDATE-BIRTHDATE)/365.25
RUN
```

```
REM Create a preprocessed dataset with categorical variable labels
REM replaced with consecutively numbered ones
REM (in same order as originals)
USE INFILE.CSV
CATEGORY OCCUPCODE DIAGNOSTIC
DISCRETE ORDER=YES
RUN SD="PREPFILE.CSV" PDM=YES
```

SCORE

Purpose

The **SCORE** command applies CART trees stored in a grove to data in your dataset, reporting prediction success tables, gains and ROC charts as well as saving predicted response(s), terminal node assignment(s) and predicted probabilities to an optional output dataset.

The command syntax is:

```
SCORE [ OFT = <yes/no>, DCM = <yes/no>,
      PROBS = <N>, PATHS = <yes/no>,
      DEPVAR = <variable> ]
```

in which the following options may be set as follows:

- | | |
|---------------|--|
| OFT | (O)mits the (F)irst (T)ree (among trees sharing a common target variable) from being a member of the committee for that target variable. When CART builds a committee of trees it also builds an "initial" tree against which the committee is compared. When scoring it may be desired for the initial tree to be added to those already in the committee. In this event, specify OFT=NO. The default is OFT=YES, consistent with previous versions of CART and the notion that the initial tree is not to be used as part of the committee. |
| DCM | (D)etails (C)ommittee (M)embers. By default, DCM=NO, in which case prediction success tables, terminal node summaries and gains and ROC charts are only produced for committees, if a committee exists in the grove. If you wish to see these reports for all trees in the committee(s), use DCM=YES. Note that DCM=YES can generate voluminous output for large committees. If no committees exist in the grove, this option is ignored and reports are printed for all trees. |
| PROBS | causes predicted probabilities (for classification models) to be added to the output dataset if there are N or fewer target classes. By default, models with five or fewer target classes will have predicted probabilities saved. |
| PATHS | causes path indicators to be added to the output dataset. By default these are not saved. |
| DEPVAR | is used to specify a proxy target (dependent) variable with a different name than the target variable used when the model was created. |

If a variable with the same name as the original target is present, or if a proxy target is specified with the DEPVAR option, SCORE will also produce misclassification or error rate reports. If the SAVE command is issued prior to SCORE, model scores will be saved to a dataset. To include all model variables in the save file, use the "/MODEL" option on the SAVE command. Merge variables may be included in the SAVE dataset by issuing the IDVAR command prior to the SCORE command. The IDVARs may be any variables on the USE dataset. The MEANS, PREDICTION, GAINS and ROC options on the LOPTIONS command will generate additional scoring output.

Examples:

```
USE "gymtutor.csv"
SAVE "testPRED.CSV" / MODEL
GROVE "BUILD_GYMc.GRV"
SCORE DEPVAR = SEGMENT, PATH = YES, PROBS = 3
```


SAVE

Purpose

The **SAVE** command saves subsequent results to a dataset. If you specify a path name, enclose the whole thing in single or double quotation marks. If an unquoted name is given without an extension, a Systat dataset is saved to the default directory and ".SYS" is appended to the name.

The command syntax is:

```
SAVE <file> [/SINGLE | DOUBLE, '<comment>']
```

Examples:

```
SAVE "/projects/scoring/Modella.csv"
```

```
SAVE "results.sas7bdat"
```

```
SAVE "/projects/scoring/Modella.xls[xls5]" (via DBMSCOPY into a  
spreadsheet)
```

```
SAVE SCORES (Save Systat dataset SCORES.SYS into the default directory)
```

```
SAVE SCORES.CSV (Save CSV dataset SCORES.CSV into the default directory)
```

The SAVE command must appear before the command that causes data to be stored to the file, e.g., you must issue the SAVE command before the SCORE command if you wish to save the scoring results to a dataset.

SEED

Purpose

The **SEED** command allows you to set the random number seed to a certain value as well as to specify that the seed remain in effect after the tree is built. Normally, the seed is reset to 13579, 12345, 131 upon starting up CART.

The command syntax is:

```
SEED I,J,K, RETAIN | NORETAIN
```

All three values I, J, K must be given. Legal values include all whole numbers between 1 and 30000. If RETAIN is not specified, the seed will be reset to 13579, 12345, 131 after the current tree is completed.

If RETAIN is specified, the seed will keep its latest value after the tree is built.

Examples:

```
SEED 1,99,7773  
SEED RETAIN  
SEED 35,784,29954, NORETAIN
```

SELECT

Purpose

The **SELECT** command selects cases from a file for analysis. You may specify up to ten simple conditions; the data preprocessor then selects those cases in the data file that meet all the conditions (that is, the conditions are linked by logical AND). SELECT commands are processed **after** any BASIC statements, allowing selections to be made based on variables created “on the fly.”

Specify each condition as variable name, logical relation, and a constant value. The variable name must come first. The six possible logical relations are =, <>, <, >, <=, and >=. You must enclose character values in quotes. Character comparisons are case sensitive.

The command syntax is:

```
SELECT <var$> <relation> '<string$>'
```

or

```
SELECT <var> <relation> <#>
```

Examples:

```
SELECT GROUP=2
SELECT GROUP<>.
SELECT AGE>=21, AGE<65
SELECT SEX$='Female', AGE>=25
```

STRATA

Purpose

The STRATA command defines a stratification variable for DATAINFO statistics. Its syntax is:

```
STRATA <variable>
```

Examples:

```
STRATA GENDER$  
DATAINFO INCOME, AGE, POLPARTY$
```

SUBMIT

Purpose

The **SUBMIT** command lets you send a text (not binary) command file to CART for processing in batch mode. The commands are executed as if you had typed them from the keyboard. If the file of commands is in the current directory (or the directory specified with Utilities/Defaults/Path) and has a .CMD extension, you need only specify the basic file name (without the extension). Otherwise, specify a path name and the complete file name enclosed in single or double quotation marks.

The command syntax is:

```
SUBMIT <file> [/ECHO ]
```

The ECHO option displays the commands on the screen as CART reads them from the SUBMIT file.

Note that screen output is automatically scrolled when you SUBMIT commands. You can use the OUTPUT command to specify an ASCII text file to review the output that is quickly generated.

Examples:

```
SUBMIT COMMANDS      (reads from file COMMANDS.CMD in current directory)
```

```
SUBMIT ' \ANALYSES\NEWJOB.CMD '  (reads from named file)
```

```
SUBMIT JOB / ECHO     (reads JOB.CMD and displays commands on screen)
```

TRANSLATE

Purpose

The **TRANSLATE** command generates reports and splitting rules from a grove file. A grove file must be named by the GROVE command prior to using the TRANSLATE command, otherwise the most recently created grove file will be used. The OUTPUT option will direct the output from TRANSLATE to the named file.

The command syntax is:

```
TRANSLATE [ LANGUAGE = CLASSIC | SAS | C | PMML | HISTORY,
          OUTPUT = "Output file",
          VLIST = <yes/no>,
          TLIST = <yes/no>,
          DETAILS = <yes/no>,
          SURROGATES = <yes/no>,
          SMI = "SAS missing value string",
          SBE = "SAS begin label",
          SDO = "SAS done label",
          SNO = "SAS node prefix",
          STN = "SAS terminal node prefix"
```

The available languages are as follows:

SAS	Implement the model in the form of a subroutine which can be included in a SAS™ data step and called with the LINK command. At present, only single-tree models are fully supported.
CLASSIC	Print the model in much the same way it is represented in the classic text output.
C	Implement the model in the form of a C language function.
PMML	Print the model using Predictive Model Markup Language (PMML) 3.1. This is an XML-based language for representing statistical models. Again, only single-tree models are fully supported. Batteries and COMBINE models are currently represented as series of single trees.
HISTORY	List the commands executed between the time CART started and when the model or battery contained in the grove was built. This is useful for reconstructing the code required to build a particular model or battery.

Example:

```
GROVE "mygrove.grv"  
TRANSLATE LANGUAGE=SAS OUTPUT="mygrove.sas"
```

Example SAS™ data step to score data with TRANSLATE output:

```
DATA OUTLIB.Scores; *Output dataset;  
  SET INLIB.NEWData; *Input dataset;  
  *Any preprocessing statements go here. We'll create a variable;  
  AGE=(&NOW-BIRTHDATE)/365.25;  
  *Score the data;  
  LINK MODELBEGIN;  
  *Any postprocessing statements could go here;  
  RETURN; *We don't want to execute the TRANSLATE output twice;  
  %INCLUDE "mygrove.sas"; *TRANSLATE output;  
  keep ID RESPONSE PROB1 PROB2;  
  rename PROB1=PROB0 PROB2=PROB1; *Original target was a 0/1 binary;  
run;
```

USE

Purpose

The **USE** command reads data from the file you specify. You may specify the root of the filename if the file resides in the current directory (usually C:\Program Files\CART 6.0\Sample Data\, if one is running the GUI, or the directory from which CART was launched, in the case of the console), or specify the directory with Utilities/Defaults/Path (in the GUI) or the FPATH command. If you specify a path, you must provide the complete file name with the appropriate extension, and surround the whole path name/file name with single or double quotation marks.

If the file name is unquoted and given without an extension, CART will search for files with the specified root name and the following extensions, in the order given:

.SYS: Native Systat binary format

.SYD: Native Systat binary format

.CSV: Comma separated text

.TXT: Comma separated text

.DAT: Comma separated text

Thus, the command “USE SOMEDATA” would cause CART to first try to open SOMEDATA.SYS in the default directory, if it exists. Otherwise, it would next try to open SOMEDATA.SYD and if it fails, continue down the list of extension until either a file with the expected name is found or the list of extensions is exhausted.

The command syntax is:

```
USE <file>
```

Examples:

```
USE MYDATA (reads from MYDATA.SYS)
USE '\MONTHLY\SURVEY.SYS'
```


WEIGHT

Purpose

The **WEIGHT** command identifies a case-weighting variable.

The command syntax is:

`WEIGHT=<variable>`

in which <variable> is a variable present in the USE dataset. The WEIGHT variable must be numeric containing any non-negative real values—no character variables.

XYPLOT

Purpose

The **XYPLOT** command produces 2-D scatter plots, plotting one or more y variables against an x variable in separate graphs.

The command syntax is:

```
XYPLOT <yvar1> [, <yvar2> , <yvar3> ] * <xvar> [ / FULL, TICKS |  
GRID, WEIGHTED, BIG ]
```

The plot is normally a half screen high; the FULL and BIG options will increase it to a full screen (24 lines) or a full page (60 lines).

TICKS and GRID add two kinds of horizontal and vertical gridding.

WEIGHTED requests plots weighted by the WEIGHT command variable.

NORMALIZED scales the vertical axis to 0 to 1 (or -1 to 1).

Examples:

```
XYPLOT IQ*AGE / FULL, GRID  
XYPLOT LEVEL(4-7)*INCOME / NORMALIZED  
XYPLOT AGE, WAGE, INDIC*DEPVAR(2) / WEIGHTED
```

Only numerical variables may be specified.

Variable groups may be used in the XYPLOT command similarly to variable names.

Appendix IV

BASIC Programming Language

This chapter provides an overview of the built-in BASIC programming language available within CART.

BASIC Programming Language

CART, and other Salford Systems' modules, contain an integrated implementation of a complete BASIC programming language for transforming variables, creating new variables, filtering cases, and database programming. Because the programming language is directly accessible anywhere in CART, you can perform a number of database management functions without invoking the data step of another program.

The BASIC transformation language allows you to modify your input files on the fly while you are in an analysis module. Permanent copies of your changed data can be obtained with the RUN command, which does no modeling. BASIC statements are applied to the data as they are read in and before any modeling takes place, allowing variables created or modified by BASIC to be used in the same manner as unmodified variables on the input dataset.

Although this integrated version of BASIC is much more powerful than the simple variable transformation functions sometimes found in other statistical procedures, it is not meant to be a replacement for more comprehensive data steps found in general use statistics packages. At present, integrated BASIC does not permit the merging or appending of multiple files, nor does it allow processing across observations. In Salford Systems' statistical analysis packages, the programming work space for BASIC is limited and is intended for on-the-fly data modifications of 20 to 40 lines of code (though custom large work space versions will accommodate larger BASIC programs). For more complex or extensive data manipulation, we recommend you use the large workspace for BASIC in DATA (available from Salford Systems) or your preferred database management software.

The remaining BASIC help topics describe what you can do with BASIC and provide simple examples to get you started. The BASIC help topics provide formal technical definitions of the syntax.

Getting Started with BASIC Programming Language

Your BASIC program will normally consist of a series of statements that all begin with a “%” sign (the “%” sign can be omitted inside of a DATA block). These statements could comprise simple assignment statements that define new variables, conditional statements that delete selected cases, iterative loops that repeatedly execute a block of statements, and complex programs with the flow control provided by GOTO statements and line numbers. Thus, somewhere before a HOT! Command such as ESTIMATE or RUN in a Salford module, you might type:

```
% LET BESTMAN = WINNER
% IF MONTH=8 THEN LET GAMES = BEGIN
% ELSE IF MONTH>8 LET GAMES = ENDED
% LET ABODE= LOG (CABIN)
```

```
% DIM COLORS(10)
% FOR I= 1 TO 10 STEP 2
% LET COLORS(I) = Y * I
% NEXT
% IF SEX$="MALE" THEN DELETE
```

The % symbol appears only once at the beginning of each line of BASIC code; it should not be repeated anywhere else on the line. You can leave a space after the % symbol or you can start typing immediately; BASIC will accept your code either way.

Our programming language uses standard statements found in many dialects of BASIC.

BASIC: Overview of BASIC Components

LET

Assigns a value to a variable. The form of the statement is:

```
% LET variable = expression
```

IF...THEN

Evaluates a condition, and if it is true, executes the statement following the THEN. The form is:

```
% IF condition THEN statement
```

ELSE

Can immediately follow an IF...THEN statement to specify a statement to be executed when the preceding IF condition is false. The form is:

```
% IF condition THEN statement
% ELSE statement
```

Alternatively, ELSE may be combined with other IF–THEN statements:

```
% IF condition THEN statement
% ELSE IF condition THEN statement
% ELSE IF condition THEN statement
% ELSE statement
```

FOR...NEXT

Allows for the execution of the statements between the FOR statement and a subsequent NEXT statement as a block. The form of the simple FOR statement is:

```
% FOR
% statements
% NEXT
```

For example, you might execute a block of statements only if a condition is true, as in

```
%IF WINE=COUNTRY THEN FOR
%LET FIRST=CABERNET
%LET SECOND=RIESLING
%NEXT
```

When an index variable is specified on the FOR statement, the statements between the FOR and NEXT statements are looped through repeatedly while the index variable remains between its lower and upper bounds:

```
% FOR [index variable and limits]
% statements
% NEXT
```

The index variable and limits form is:

```
%FOR I= start-number TO stop-number [ STEP = stepsize ]
```

where I is an integer index variable that is increased from *start-number* to *stop-number* in increments of *stepsize*. The statements in the block are processed first with I = *start-number*, then with I = *start-number* + *stepsize*, and repeated until I >=*stop-number*. If STEP=*stepsize* is omitted, the default is to step by 1. Nested FOR-NEXT loops are not allowed.

DIM

Creates an array of subscripted variables. For example, a set of five scores could be set up with:

```
% DIM SCORE(5)
```

This creates the variables SCORE(1), SCORE(2), ..., SCORE(5).

The size of the array must be specified with a literal integer up to a maximum size of 99; variable names may not be used. You can use more than one DIM statement, but be careful not to create so many large arrays that you exceed the maximum number of variables allowed (currently 8019).

DELETE

Deletes the current case from the data set.

Operators

The table below lists the operators that can be used in BASIC statement expressions. Operators are evaluated in the order they are listed in each row with one exception: a minus sign before a number (making it a negative number) is evaluated after exponentiation and before multiplication or division. The "<>" is the "not equal" operator.

Numeric Operators	()	^	*	/	+	-
Relational Operators	<	<=	<>	=	=>	>
Logical Operators	AND	OR	NOT			

BASIC Special Variables

BASIC has five built-in variables available for every data set. You can use these variables in BASIC statements and create new variables from them. You may not redefine them or change their values directly.

<u>Variable</u>	<u>Definition</u>	<u>Values</u>
<u>CASE</u>	observation number	1 to maximum observation number
<u>BOF</u>	logical variable for beginning of file	1 for first record in file, 0 otherwise
<u>EOF</u>	logical variable for end of file	1 for last record in file, 0 otherwise
<u>BOG</u>	logical variable for beginning of BY group	1 for first record in BY group, 0 otherwise
<u>EOG</u>	logical variable for end of BY group	1 for last record in BY group, 0 otherwise

BY groups are not supported in CART, so BOG and EOG are synonymous with BOF and EOF.

BASIC Mathematical Functions

Integrated BASIC also has a number of mathematical and statistical functions. The statistical functions can take several variables as arguments and automatically adjust for missing values. Only numeric variables may be used as arguments. The general form of the function is:

```
FUNCTION(variable, variable, ...)
```

Integrated BASIC also includes a collection of probability functions that can be used to determine probabilities and confidence level critical values, and to generate random numbers.

Multiple-Argument Functions

Function	Definition	Example
AVG	arithmetic mean	%LET XMEAN=AVG(X1,X2,X3)
MAX	maximum	%LET BEST=MAX(Y1,Y2,Y3,Y4,Y5)
MIN	minimum	%LET MINCOST=MIN(PRICE1,OLDPRICE)
MIS	number of missing values	
STD	standard deviation	
SUM	summation	

Single-Argument Functions

Function	Definition	Example
ABS	absolute value	%ABSVAL=ABS(X)
ACS	arc cosine	
ASN	arc sine	
ATH	arc hyperbolic tangent	
ATN	arc tangent	
COS	cosine	
EXP	exponential	
LOG	natural logarithm	%LET LOGXY=LOG(X+Y)
SIN	sine	
SQR	square root	%LET PRICESR=SQR(PRICE)
TAN	tangent	

The following shows the distributions and any parameters that are needed to obtain values for either the random draw, the cumulative distribution, the density function, or the inverse density function. Every function name is composed of three letters:

Key-Letter:

This first letter identifies the distribution.

Distribution-Type Letters:

RN (random number), CF (cumulative),
DF (density), IF (inverse).

BASIC Probability Functions

CART BASIC also includes a collection of probability functions that can be used to determine probabilities and confidence level critical values, and to generate random numbers.

The following table shows the distributions and any parameters that are needed to obtain values for the random draw, the cumulative distribution, the density function, or the inverse density function. Every function name is composed of two parts:

The "Key" (first) letter identifies the distribution.

Remaining letters define function: RN (random number), CF (cumulative), DF (density), IF (inverse).

Distribution	Key-Letter	Random Draw (RN)	Cumulative (C) Density (D) Inverse (I)	Comments (α is the probability for inverse density function)
Beta	B	BRN	BCF(β ,p,q) BDF(β ,p,q) BIF(α ,p,q)	β = beta value p,q = beta parameters
Binomial	N	NRN(n,p)	NCF(x,n,p) NDF(x,n,p) NIF(a,n,p)	n = number of trials p = prob of success in trial x = binomial count
Chi-square	X	XRN(df)	XCF(χ^2 ,df) XDF(χ^2 ,df) XIF(α ,df)	χ^2 = chi-squared valued f = degrees of freedom
Exponential	E	ERN	ECF(x) EDF(x)EIF(a)	x = exponential value
F	F	FRN(df1,df2)	FCF(F,df1,df2) FDF(F,df1,df2) FIF(α ,df1,df2)	df1, df2 = degrees of freedom F = F-value
Gamma	G	GRN(p)	GCF(γ ,p) GDF(γ ,p) GIF(α ,p)	p = shape parameter γ = gamma value
Logistic	L	LRN	LCF(x) LDF(x) LIF(α)	x = logistic value
Normal (Standard)	Z	ZRN	ZCF(z)	z = normal z-score

			ZDF(z) ZIF(a)	
Poisson	P	PRN(p)	PCF(x,p) PDF(x,p) PIF(α ,p)	p = Poisson parameter x = Poisson value
Studentized	S	SRN(k,df)	SCF(s,k,df) SDF(s,k,df) SIF(α ,k,df)	k = parameter f = degrees of freedom
t	T	TRN(df)	TCF(t,df) TDF(t,df) TIF(α ,df)	df = degrees of freedom t = t-statistic
Uniform	U	URN	UCF(x) UDF(x) UIF(α)	x = uniform value
Weibull	W	WRN(p,q)	WCF(x,p,q) WDF(x,p,q) WIF(α ,p,q)	p = scale parameter q = shape parameter

These functions are invoked with either 0, 1, or 2 arguments as indicated in the table above, and return a single number, which is either a random draw, a cumulative probability, a probability density, or a critical value for the distribution.

We illustrate the use of these functions with the chi-square distribution. To generate 10 random draws from a chi-square distribution with 35 degrees of freedom for each case in your data set:

```
% DIM CHISQ(10)
% FOR I= 1 TO 10
% LET CHISQ(I)=XRN(35)
% NEXT
```

To evaluate the probability that a chi-square variable with 20 degrees of freedom exceeds 27.5:

```
%LET CHITAIL=1 - XCF(27.5, 20)
```

The chi-square density for the same chi-square value is obtained with:

```
%LET CHIDEN=XDF(27.5, 20)
```

Finally, the 5% point of the chi-squared distribution with 20 degrees of freedom is calculated with:

```
%LET CHICRIT=XIF(.95, 20)
```

Missing Values

The system missing value is stored internally as the largest negative number allowed. Missing values in BASIC programs and printed output are represented with a period or dot ("."), and missing values can be generated and their values tested using standard expressions.

Thus, you might type:

```
%IF NOSE=LONG THEN LET ANSWER=.  
%IF STATUS=. THEN DELETE
```

Missing values are propagated so that most expressions involving variables that have missing values will themselves yield missing values.

One important fact to note: because the missing value is technically a very large negative number, the expression $X < 0$ will evaluate as true if X is missing.

BASIC statements included in your command stream are executed when a HOT! Command such as ESTIMATE, APPLY, or RUN is encountered; thus, they are processed before any estimation or tree building is attempted. This means that any new variables created in BASIC are available for use in MODEL and KEEP statements, and any cases that are deleted via BASIC will not be used in the analysis.

More Examples

It is easy to create new variables or change old variables using BASIC. The simplest statements create a new variable from other variables already in the data set. For example:

```
% LETPROFIT=PRICE *QUANTITY2* LOG(SQFTRENT) , 5*SQR(QUANTITY)
```

BASIC allows for easy construction of Boolean variables, which take a value of 1 if true and 0 if false. In the following statement, the variable XYZ would have a value of 1 if any condition on the right-hand side is true, and 0 otherwise.

```
% LET XYZ = X1<.5 OR X2>17 OR X3=6
```

Suppose your data set contains variables for gender and age, and you want to create a categorical variable with levels for male-senior, female-senior, male-non-senior, female-non-senior. You might type:

```
% IF MALE = . OR AGE = . THEN LET NEWVAR = .
% ELSE IF MALE = 1 AND AGE < 65 THEN LET NEWVAR=1
% ELSE IF MALE = 1 AND AGE >= 65 THEN LET NEWVAR=2
% ELSE IF MALE = 0 AND AGE < 65 THEN LET NEWVAR=3
% ELSE LET NEWVAR = 4
```

If the measurement of several variables changed in the middle of the data period, conversions can be easily made with the following:

```
% IF YEAR > 1986 OR MEASTYPE$="OLD" THEN FOR
% LET TEMP = (OLDTEMP-32)/1.80
% LET DIST = OLDDIST / .621
% NEXT
% ELSE FOR
% LET TEMP = OLDTEMP
% LET DIST = OLDDIST
% NEXT
```

If you would like to create powers of a variable (square, cube, etc.) as independent variables in a polynomial regression, you could type something like:

```
% DIM AGEPOWER(5)
% FOR I = 1 TO 5
% LET AGEPOWER(I) = AGE^I
% NEXT
```

Filtering the Data Set or Splitting the Data Set

Integrated BASIC can be used for flexibly filtering observations. To remove observations with SSN missing, try:

```
% IF SSN= . THEN DELETE
```

To delete the first 10 observations, type:

```
% IF CASE <= 10 THEN DELETE
```

Because you can construct complex Boolean expressions with BASIC, using programming logic combined with the DELETE statement gives you far more control than is available with the simple SELECT statement. For example:

```
% IF AGE>50 OR INCOME<15000 OR (REGION=9 AND GOLF=.) THEN DELETE
```

It is often useful to draw a random sample from a data set to fit a problem into memory or to speed up a preliminary analysis. By using the uniform random number generator in BASIC, this is easily accomplished with a one-line statement:

```
% IF URN < .5 THEN DELETE
```

The data set can be divided into an analysis portion and a separate test portion distinguished by the variable TEST:

```
% LET TEST= URN < .4
```

This sets TEST equal to 1 in approximately 40% of all cases and 0 in all other cases. The following draws a stratified random sample taking 10% of the first stratum and 50% of all other strata:

```
% IF DEPVAR = 1 AND URN < .1 THEN DELETE
% ELSE IF DEPVAR<>1 AND URN < .5 THEN DELETE
```

DATA Blocks

A DATA block is a block of statements appearing between a DATA command and a DATA END command. These statements are treated as BASIC statements, even though they do not start with “%.” Here is an example:

```
DATA
let ranbeta1=brn(.25,.75)
let ranbeta2=brn(.75,.25)
let ranbin1=nrn(100,.25)
let ranbin2=nrn(500,.75)
let ranchi1=xrn(1)
let ranchi2=xrn(2)
DATA END
```

Advanced Programming Features

Integrated BASIC also allows statements to have line numbers that facilitate the use of flow control with GOTO statements. Line numbers must be integers less than 32000, and we recommend that if you use any line numbers at all, all your BASIC statements should be numbered. BASIC will execute the numbered statements in the order of the line numbers, regardless of the order in which the statements are typed, and unnumbered BASIC statements are executed before numbered statements.

Here is an example of using the GOTO:

```
%10 IF PARTY=GOP THEN GOTO 96
%20 LET NEWDEM=1
%30 LET VEEP$="GORE"
%40 GOTO 99
%96 LET VEEP$="KEMP"
%99 LET CAMPAIGN=1
```

BASIC Programming Language Commands

The following pages contain a summary of the BASIC programming language commands. They include syntax usage and examples.

DELETE Statement

Purpose

Drops the current case from the data set.

Syntax

```
% DELETE  
% IF condition THEN DELETE
```

Examples

To keep a random sample of 75% of a data set for analysis:

```
% IF URN < .25 THEN DELETE
```

DIM Statement

Purpose

Creates an array of subscripted variables.

Syntax

```
% DIM var(n)
```

where *n* is a literal integer. Variables of the array are then referenced by variable name and subscript, such as *var*(1), *var*(2), etc.

In an expression, the subscript can be another variable, allowing these array variables to be used in FOR...NEXT loop processing. See the section on the FOR...NEXT statement for more information.

Examples

```
% DIM QUARTER(4)
% DIM MONTH(12)
% DIM REGION(9)
```

ELSE Statement

Purpose

Follows an IF...THEN to specify statements to be executed when the condition following a preceding IF is false.

Syntax

The simplest form is:

```
% IF condition THEN statement1
% ELSE statement2
```

The statement2 can be another IF...THEN condition, thus allowing IF...THEN statements to be linked into more complicated structures. For more information see the section for IF...THEN.

Examples

```
% 5 IF TRUE=1 THEN GOTO 20
% 10 ELSE GOTO 30
% IF AGE <=2 THEN LET AGEDES$ = "baby"
% ELSE IF AGE <= 18 THEN LET AGEDES$ = "child"
% ELSE IF AGE < 65 THEN LET AGEDES$ = "adult"
% ELSE LET AGEDES$ = "senior"
```


FOR...NEXT Statement

Purpose

Allows the processing of steps between the FOR statement and an associated NEXT statement as a block. When an optional index variable is specified, the statements are looped through repetitively while the value of the index variable is in a specified range.

Syntax

The form is:

```
% FOR [index variable and limits]
% statements
% NEXT
```

The index variable and limits is optional, but if used, it is of the form

```
x = y TO z [STEP=s]
```

where x is an index variable that is increased from y to z in increments of s. The statements are processed first with $x = y$, then with $x = y + s$, and so on until $x = z$. If STEP=s is omitted, the default is to step by 1.

Remarks

Nested FOR...NEXT loops are not allowed and a GOTO which is external to the loop may not refer to a line within the FOR...NEXT loop. However, GOTOs may be used to leave a FOR...NEXT loop or to jump from one line in the loop to another within the same loop.

Examples

To have an IF...THEN statement execute more than one statement if it is true:

```
% IF X<15 THEN FOR
% LET Y=X+4
% LET Z=X-2
% NEXT
```

GOTO Statement

Purpose

Jumps to a specified numbered line in the BASIC program.

Syntax

The form for the statement is:

```
% GOTO ##
```

where **##** is a line number within the BASIC program.

Remarks

This is often used with an IF...THEN statement to allow certain statements to be executed only if a condition is met.

If line numbers are used in a BASIC program, all lines of the program should have a line number. Line numbers must be positive integers less than 32000.

Examples

```
% 10 GOTO 20
% 20 STOP
% 10 IF X=. THEN GOTO 40
% 20 LET Z=X*2
% 30 GOTO 50
% 40 LET Z=0
% 50 STOP
```

IF . . . THEN Statement

Purpose

Evaluates a condition and, if it is true, executes the statement following the THEN.

Syntax

```
% IF condition THEN statement
```

An IF...THEN may be combined with an ELSE statement in two ways. First, the ELSE may be simply used to provide an alternative statement when the condition is not true:

```
% IF condition THEN statement1  
% ELSE statement2
```

Second, the ELSE may be combined with an IF...THEN to link conditions:

```
% IF condition THEN statement  
% ELSE IF condition2 THEN statement2
```

To allow multiple statements to be conditionally executed, combine the IF...THEN with a FOR...NEXT:

```
% IF condition THEN FOR  
% statement  
% statement  
% NEXT
```

Examples

To remove outlier cases from the data set:

```
% IF ZCF (ABS ((z-zmean)/zstd)) > .95 THEN DELETE
```

LET Statement

Purpose

Assign a value to a variable.

Syntax

The form of the statement is:

```
% LET variable = expression
```

The expression can be any mathematical expression, or a logical Boolean expression. If the expression is Boolean, then the variable defined will take a value of 1 if the expression is true, or 0 if it is false. The expression may also contain logical operators such as AND, OR and NOT.

Examples

```
% LET AGEMONTH = YEAR - BYEAR + 12*(MONTH , BMONTH)
% LET SUCCESS =(MYSPEED = MAXSPEED)
% LET COMPLETE = (OVER = 1 OR END=1)
```

STOP Statement

Purpose

Stops the processing of the BASIC program on the current observation. The observation is kept but any BASIC statements following the STOP are not executed.

Syntax

The form of the statement is:

```
% STOP
```

Examples

```
%10 IF X = 10 THEN GOTO 40  
%20 ELSE STOP  
%40 LET X = 15
```


Bibliography

Breiman, L. (1996). *Arcing classifiers* (Technical Report). Berkeley: Statistics Department, University of California.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24, 123-140.

Breiman, Leo, Jerome Friedman, Richard Olshen, and Charles Stone. (1984) *Classification and Regression Trees*. Pacific Grove: Wadsworth.

Dietterich, T. (1998). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40, 139-158.

Freund, Y. & R. E. Schapire. (1996). Experiments with a new boosting algorithm. In L. Saitta, ed., *Machine Learning: Proceedings of the Thirteenth National Conference*, Morgan Kaufmann, pp. 148-156.

Steinberg, Dan and Phillip Colla. (1997) *CART--Classification and Regression Trees*. San Diego, CA: Salford Systems.

Index

[Buttons]

- [...] button, 132
- [-] button, 57, 241, 293
- [+] button, 57, 241, 293
- [<-Send To Left] button, 273
- [...] button, 142
- [1 SE] button, 201, 204
- [Add to List] button, 100
- [Add] button, 200
- [Advanced] button, 122
- [All Classes] button, 63, 248
- [All] button, 159
- [Apply] button, 57, 59, 242
- [Ave. Profit] button, 152
- [Average] button, 208
- [Bar] button, 201
- [Bars] button, 197
- [Both] button, 62, 246
- [Box Plot] button, 205
- [Brief] button, 292
- [CART] button, 234
- [Cases] button, 71, 254
- [Change...] button, 270, 273
- [Chart] button, 208
- [Color...] button, 59, 242
- [Column %] button, 177, 250
- [Columns] button, 196
- [COMBINE] button, 162
- [Continue] button, 26, 101, 125
- [Copy to Internal Nodes] button, 58, 241
- [Copy to Terminal Nodes] button, 58, 241
- [Copy] button, 26, 27
- [Cum Lift] button, 143
- [Cum. Ave. Profit] button, 153
- [Cum. Profit] button, 153
- [Defaults] button, 117, 287
- [Delete from List] button, 100
- [Filtering] button, 197
- [Full] button, 292
- [Fuzzy Match] button, 190
- [Gains] button, 143
- [Grid] button, 205
- [Grove...] button, 171, 237, 257
- [Grow] button, 56, 236, 244
- [Larger] button, 75, 237
- [Learn] button, 62, 71, 75, 151, 153, 183, 197, 201, 204, 237, 246, 253, 256, 259
- [Legend] button, 204
- [Lift] button, 143
- [Line] button, 201
- [Max] button, 205, 208
- [Mean] button, 205
- [Median] button, 205
- [Merge selected groups] button, 138
- [Min Cost] button, 201, 203, 204
- [Min] button, 205, 208
- [Misclass] button, 201, 204
- [Model...] button, 83, 146
- [Next Prune] button, 54, 236
- [Nodes] button, 201
- [None] button, 208
- [Open] button, 43, 140, 230, 298
- [Optimal Tree] button, 174
- [Optimal Tree] button, 182
- [Other Classes] button, 63, 247
- [Page Setup...] button, 60, 141, 243
- [Pct] button, 71, 254
- [Pooled] button, 183, 256, 259
- [Profit] button, 152
- [Prune] button, 56, 236, 244
- [Quartile 0.25] button, 205
- [Quartile 0.75] button, 205
- [Recall Defaults] button, 127
- [Rel. Error] button, 201
- [Report Now] button, 289
- [ROC] button, 143, 201, 204
- [Row %] button, 177, 250
- [Save as Defaults] button, 127
- [Save Grove] button, 202
- [Save Grove...] button, 75, 78, 87, 166, 171, 259
- [Save Navigator...] button, 74
- [Save] button, 74, 140, 171, 257, 283, 284
- [Scatter] button, 197
- [Score...] button, 75, 78, 172, 178, 237, 259
- [Select Variables] button, 110
- [Select] button, 174
- [Select...] button, 78, 173, 178, 181, 182, 259, 260
- [Send To Right->] button, 273
- [Set Class Names] button, 92
- [Set Default] button, 288
- [Set Defaults] button, 57, 58, 241, 242
- [Set Focus Class...] button, 91
- [Set Left] button, 273
- [Set Right] button, 273
- [Set Root] button, 268
- [Show Min Error] button, 201

[Show] button, 197
 [Smaller] button, 75, 237
 [Sorting] button, 197
 [Split selected groups] button, 139
 [Splitters...] button, 51, 151, 238
 [Start] button, 26, 47, 135, 149, 166, 171, 200, 234, 269, 270, 274, 298
 [Summary Reports...] button, 61, 143, 151, 186, 244
 [Symmetrical] button, 117
 [T/T Consist...] button, 187
 [Table] button, 208
 [Tagged] button, 159
 [Test] button, 62, 71, 75, 151, 153, 183, 197, 201, 204, 237, 246, 250, 253, 256, 259
 [Translate...] button, 75, 77, 180, 237, 259
 [Tree Details...] button, 56, 141, 151, 239
 [Tree Details...] button, 271, 274
 [Unlock] button, 27
 [Use Default] button, 288
 [View Data] button, 44

A

above depth, 278
 accessing data, 31, 34
 Accuracy tab, 202
 1 SE Terminal Nodes, 203
 Average Accuracy, 203
 Avg. ROC, 203
 Class Accuracy, 203
 Class ROC, 203
 Opt. Terminal Nodes, 203
 Overall Accuracy, 204
 Rel. Error, 203
 Activity Window, 18, 146
 ADJUST command, 328
 advanced options, 111
 advanced programming features, 415
 advanced settings, 147, 148
 Advanced tab, 85, 147, 148, 232, 233
 ARCing, 162, 163
 power setting, 164
 ASCII files, 33
 character variables, 33
 numeric variables, 33
 association, 248, 252
 auto validation, 19
 AUXILIARY command, 329
 auxiliary variables, 85, 91
 color coding, 136
 merge selected groups, 138
 split selected groups, 139
 viewing information, 134
 Averaging tab, 208

B

bagging, 162
 bar chart, 247
 BASIC
 data management, 14
 mathematical functions, 410
 probability functions, 411
 programming language commands, 416
 programming language overview, 406
 special variables, 409
 BASIC programming language, 297, 298
 batch command file, 297
 batch processing, 296, 298
 Batteries, 16, 200
 CV, 206
 CVR, 208
 DEPTH, 210
 DRAW, 210
 FLIP, 211
 KEEP, 212
 LOVO, 214
 MCT, 215
 MINCHILD, 216
 MVI, 216
 NODES, 218
 ONEOFF, 218
 PRIOR, 219
 RULES, 220
 SAMPLE, 221
 SHAVING, 222
 SUBSAMPLE, 223
 TARGET, 224
 battery, 147
 battery models, 194
 Battery Options, 200
 Battery Summary
 1 SE Terminal Nodes, 201
 Accuracy tab, 202
 Avg. ROC, 202
 Classification Battery Models, 201
 Contents tab, 202
 Error Profiles tab, 204
 Model Name, 201
 Model Specifications, 202
 Opt. Terminal Nodes, 201
 Rel. Error, 202
 Var. Imp. Averaging tab, 208
 Var. Imp. tab, 204
 Battery tab, 147
 Battery Types, 200
 beginning of file, 409
 beginning of group, 409
 below depth, 278
 best tree, 102, 146
 Best Tree tab, 85, 146, 147, 232, 233
 binary-split, 11
 bootstrap resampling, 163

- Bootstrapping, 162
- BOPTION command
 - MISSING, 115
- BOPTIONS command, 95, 104, 115, 334
 - BRIEF, 132
 - COMPETITORS, 131
 - COMPLEXITY, 112
 - COPIOUS, 132
 - CVLEARN, 113
 - PRINT, 130
 - TREELIST, 131
- Boston Housing data, 146
- box plots, 156
- Box Plots tab, 156
- BUILD command, 338
- building trees
 - classification, 82
 - regression, 146

C

- CART monograph, 11, 20
- CART Notepad, 298
- CART Report window, 178
- case weights, 90
 - missing, 90
 - negative values, 90
 - zeroed, 90
- Categorical tab, 146, 232
- categorical variables, 47, 88, 92, 146, 234
 - high-level, 94
- CATEGORY command, 339
- CDF command, 340
- character variable names, 33, 35
- character variables, 33, 89
- CHARSET command, 341
- Chart Type, 153
- child node, 139, 253
- class assignment, 238
- Class Assignment dialog, 59, 242
- CLASS command, 91, 93, 109, 342
- class names, 59, 92, 242
- class probability, 13, 105
- Classic rules, 159
- classification trees, 82
- cluster analysis, 264
- color coding, 57, 59, 151, 235, 241, 242
 - auxiliary variables, 136
 - tagged nodes, 158
- COMBINE command, 344
- combine controls, 165
 - number of sample redraws, 165
 - number of trees, 165
- combine method, 164
- comma delimited, 37
- command file
 - ATOM.CMD, 200
 - CLASS.CMD, 302, 308, 309

- CLASSCOMB.CMD, 309
- CMD, 79, 93, 261
- CV.CMD, 206
- CVR.CMD, 208
- DEPTH.CMD, 210
- DRAW.CMD, 211
- FLIP.CMD, 211
- HOTSPOT.CMD, 194
- KEEP.CMD, 212
- LOVO.CMD, 214
- MCT.CMD, 215
- MVI.CMD, 216
- ONEOFF.CMD, 218
- PRIORS.CMD, 219
- REG.CMD, 305
- RULES.CMD, 220
- SAMPLE.CMD, 221
- SHAVING.CMD, 223
- TARGET.CMD, 224
- TTC.CMD, 186
- command file (*.cmd), 297
- command input, 298
- command line equivalents, 315
- command log, 78, 261, 297, 299, 300
- command prompt, 298
- command reference, 327
- command sequence, 300
- command syntax, 300, 301
 - classification example, 302
 - committee tree (combine) example, 308
 - regression example, 305
 - scoring example, 309
- command-line, 296
- command-line mode, 298
- committee of experts
 - ARCing, 163
 - bootstrap resampling, 163
 - combine controls, 165
 - combine method, 164
 - evaluation sample holdout, 165
 - files to save, 166
 - pruning test method, 165
 - report details, 166
 - specify model, 164
- committee tree, 167
- comparing child nodes, 139
- comparing learn and test, 139
- competitors, 68, 155, 251
 - number to report, 131
- Competitors and Surrogates tab, 155
- complexity parameter, 112
- confusion matrix, 250
- Consistency by Trees, 188
 - Dir. Fail Count, 189
 - Direction Max Z, 189
 - Directional Agreement, 188
 - Rank Fail Count, 189
 - Rank Match, 188

- Rank Max Z, 189
- table, 191
- Terminal Nodes, 188
- Tree Name, 188
- Consistency Details by Nodes, 189
- Lift Learn, 189
- Lift Test, 189
- N Focus Learn, 189
- N Focus Test, 189
- N Node Learn, 189
- N Node Test, 189
- N Other Learn, 189
- N Other Test, 189
- constrains
 - predictor groups, 277
- constraints, 15, 146
 - learn sample, 281
- Constraints tab, 275
- Contents tab, 202
 - Model Specifications, 202
- continuous target variables, 146
- Contraints tab, 146
- contribution
 - variable, 248
- control modes, 297
- converting older tree files, 172
- copy, 284
- co-relational analysis, 218
- correlation structure, 267
- cost matrix, 116
- Cost tab, 233
- costs, 12
- Costs tab, 85
- counts
 - comparing learn/test, 139
- covariance matrix, 224
- creating batch files, 298
- creating new variables, 101
- cross validation, 12, 18, 96, 206, 208
 - data size warning, 112
 - reporting options, 131
- CSV, 37
- cut, 290
- D**
- data
 - accessing, 32
 - ASCII, 32
 - DBMS/COPY, 32
 - methods of reading, 32
 - SPAMBASE.CSV, 186
- data files
 - BOSTON.CSV, 146, 178, 206, 218
 - FNCELLA.CSV, 216
 - GOODBAD.CSV, 40, 43, 134, 183
 - GYMTUTOR.CSV, 134, 171, 176, 208, 228, 267, 277, 279
 - HOSLEM.CSV, 82, 91
 - PROSTATE2.CSV, 220
 - SAMPLE.CSV, 34
 - SPAM.CSV, 194
 - SPAMBASE.CSV, 210, 211, 212, 214, 215, 219, 221, 223, 224
- data information, 291
 - descriptive statistics, 293
 - extreme values, 292
 - frequency tables, 291, 292
 - include variables, 292
 - location, 293
 - maximum levels, 292
 - maximum tabulations, 292
 - quantiles, 293
 - saving to grove, 292
 - strata variable, 292
 - variability, 293
 - weight variable, 292
- data management, 14
- data preparation, 14
- data viewer, 290
- DATAINFO command, 294, 346
- DBMS/COPY
 - ASCII format, 33
 - Excel format, 36
- default directories, 29, 132
- default display setting, 57, 241
- default settings, 85, 147
- DELETE command, 409, 416
- delimited text
 - comma, 33
 - semicolon, 33
 - spaces, 33
 - tabs, 33
- dependent variable, 85
- depth, 210
- depth of tree, 113, 287
- DESCRIPTIVE command, 347
- descriptive statistics, 15, 291
- Desktop, 41, 228
- detailed node report, 238
- DIM command, 408, 417
- directional instability, 187
- directional stability, 187
- directories, 29, 134
 - input files, 133
 - output files, 133
 - specify defaults, 132
 - temporary files, 133
 - user specified, 28
- Directories tab, 28
 - control functions, 29
 - Input files, 28, 133
 - Output files, 28, 133
 - Temporary files, 29, 133
- disallow, 275
- DISALLOW command, 282

discount surrogates, 103, 248
DISCRETE command, 115, 348
MISSING, 115
Display Tree, 242
displaying tree rules, 76, 258

E

Edit menu, 229
Copy, 284
Fonts..., 76, 258
Options..., 125
effective frontier, 198
ELSE command, 407, 418
embedded grove information, 172, 180
embedded model information, 171, 172, 180
end of file, 409
end of group, 409
ensemble of trees, 162
entropy, 13, 105
ERROR command, 352
Error Profiles tab, 204
error rate, 66, 67, 249, 250
errors and warnings, 319
evaluation sample holdout, 165
even splits, 106
Excel format, 36
EXCLUDE command, 353
exploratory tree, 96
exporting tree rules, 76, 183, 258

F

file formats, 34
File menu, 42, 43, 229, 230, 284
Command Prompt, 298
Export..., 183, 259
Log Results to..., 283
most recently used file, 132
New Notepad..., 284, 300
Open>Command File..., 300
Open>Data File..., 34
Page Setup..., 144
Print Preview..., 290
Print Setup..., 290
Print..., 60, 141, 144, 243, 284, 290
Save, 140, 300
Save As..., 284
Save CART Output..., 76, 258
Save CART Output..., 284
Save Grove..., 257
Save Navigator..., 74, 140
Submit Command File, 298, 301
Submit Current Line to End, 299
Submit Window, 299, 300
File of type:, 34, 37
files
.TR1, 172

grove, 170, 172
navigator, 170
flat file, 32
focus class, 91
fonts, 76, 258
FOR...NEXT command, 408, 419
FORCE command, 275
Force Split tab, 146
Force Splits tab, 267
forced splits, 15, 146
FORMAT command, 126, 356
fraction of cases for testing, 98
frequency distribution, 136
fuzzy match, 187, 190

G

gains chart, 18, 61, 245
overlying, 143
printing, 143
gini, 13, 105
GOTO command, 420
GROUP command, 357
GROVE command, 141, 171, 172, 180, 182, 358
grove files, 170
grove information
embedded, 172
groves, 14
growing tree, 47, 234, 236

H

HARVEST command, 180, 359
HELP command, 361
Help menu, 42, 229
high level categorical, 88
high level categorical penalty, 121, 124
high level categorical predictors, 94
HISTOGRAM command, 362
Hot Spot Detection, 17, 194
Hot Spots, 194
Hotspot Chart, 195, 197
Hotspot Setup, 195
Hotspot Table, 195
Edit Spread, 196, 197
Learn Richness, 196
Learn Sample Count, 196
Node, 196
Test Richness, 196
Test Sample Count, 196
Tree, 196
hyper-link, 257

I

icons, 42
 IDVAR command, 363
 IF...THEN command, 407, 421
 improvement, 69, 252
 indicators
 missing values, 34
 initial tree, 167
 input files
 default directory, 28, 133
 installation
 custom, 25
 permissions, 26
 procedure, 25
 typical, 25
 introduction, 10

K

KEEP command, 212, 364
 keyboard conventions, 43
 keyboard shortcuts, 43

L

LABEL command, 365
 labels
 assigning, 59, 242
 language
 command-line, 296
 learn sample, 210, 211, 281
 learn sample size, 113, 287
 least absolute deviation, 13, 147
 least squares, 13, 147
 LET command, 407, 422
 level of detail, 238
 lift index, 246
 LIMIT command, 112, 287, 369
 ATOM, 111
 DEPTH, 113
 LEARN, 114
 MINCHILD, 111
 NODES, 113
 limits
 specifying growth size, 113, 286
 linear combinations, 17, 104, 108
 estimating number of splits, 108
 LC lists, 17
 minimum node sample size, 108
 selected variables, 109
 variable deletion, 108
 LINEAR command, 371
 logical operators, 409
 LOPTIONS command, 372
 PRINT, 126

M

main splitters, 238
 main tree, 56, 57, 239, 241
 main tree rules, 159
 Max Cases, 281
 MEMORY command, 374
 memory management, 285
 memory problems, 74, 257
 memory requirements, 285
 memory usage example, 286
 menus, 41, 229
 method. See splitting rules
 METHOD command, 375
 Method tab, 85, 146, 147, 232, 233
 methodology, 10
 Min Cases, 281
 MINCHILD command, 216
 minimum cost tree, 102, 103
 MISCLASS command, 376
 misclassification, 12, 67, 250
 misclassification costs, 116
 misclassification table, 61, 66, 245, 249
 MISCLASSIFY command, 118
 missing case weight, 90
 missing value analysis, 15, 114
 missing value controls, 15, 114
 missing value indicators, 15, 114
 missing values, 12, 34, 77, 84, 103, 147, 259,
 290, 323, 348, 386, 410, 413
 penalty, 121, 123, 216
 missing values indicators, 216
 model automation, 16, 200
 MODEL command, 377
 model information, 175
 embedded, 172
 Model menu, 42, 229
 Construct Model..., 78, 261
 model setup
 classification trees, 83
 default settings, 85, 147
 regression trees, 146
 setting limits, 113, 286
 Model Setup, 84, 129, 146
 Advanced tab, 216
 Advanced tab, 147, 148
 Battery tab, 147, 200
 Best Tree tab, 146
 Categorical tab, 146
 Constraints tab, 146
 Force Split tab, 146
 Method tab, 146
 Model tab, 146, 147
 Penalty tab, 147, 216
 Select Cases tab, 146
 Testing tab, 210, 211
 Model Setup dialog, 34, 45, 78, 83, 234
 Advanced tab, 85, 111, 232, 233

- Best Tree tab, 85, 102, 232, 233
- Categorical tab, 92, 94, 232
- Combine tab, 164
- Constraints tab, 275
- Cost tab, 116, 233
- Costs tab, 85
- Force Splits tab, 267
- Method tab, 85, 104, 232, 233
- Model tab, 85, 232, 233, 264, 266
- Penalty tab, 85, 121, 233
- Priors tab, 85, 118, 233
- Select Cases tab, 100, 232
- Testing tab, 95, 232, 233
- model specifications
 - saving, 140
- Model tab, 85, 146, 147, 232, 233, 266
- model translation, 20, 180
- models
 - scoring, 170, 172, 173
 - translating, 170, 180
- Monte Carlo test, 19, 215
- MOPTIONS command, 378
- MRU files (most recently used), 29, 132
- MVI, 15, 114, 216

N

- NAMES command, 380
- Navigator window, 48, 149, 158, 235
- navigators, 14, 61, 170, 171, 244
 - opening, 140
 - saving, 140
- negative case weight, 90
- NEW command, 381
- no independent testing, 96
- node assignment, 175
- Node Detail..., 57, 241, 242
- Node Display, 238
- node frequency distributions, 71, 253
- node report, 158
- Node Report window, 68, 154, 251
- Node Reports
 - Box Plots tab, 156
 - Classification tab, 71, 253
 - Competitors and Surrogates tab, 68, 155, 251
 - Rules tab, 72, 156, 255
 - Splitter tab, 73, 157, 254
- node size, 275
- node split, 238
- node statistics, 158
- nodes, 287
 - comparing children, 139
 - comparing learn/test, 139
 - maximum number, 113, 287
 - parent node minimum cases, 111
 - richnes, 247
 - terminal node minimum size, 111

- NODES command, 218
- node-specific median, 156
- non-linearities, 219, 224
- NOTE command, 382
- notepad, 300
- number of surrogates, 103
- number of variables, 32
- numeric operators, 409

O

- observation number, 409
- Open Data File, 34, 35
- Open File icon, 43, 230
- Open>Data File...
 - File menu, 34
- Open..., 43, 230
- opening
 - file, 43, 230
 - navigators, 140
- operators
 - logical, 409
 - numeric, 409
 - relational, 409
- opt, 128
- optimal models, 186
- optimal tree, 102
- options, 125
 - advanced, 111
 - classic output, 127
 - command notation, 127
 - default display window, 127
 - Directories tab, 28, 132
 - Random Number tab, 132
 - Report Writer, 289
 - Reporting tab, 129
 - ROC graph labels, 127
 - text reports, 125
- OPTIONS command, 383
- Options dialog, 125
- ordered twoing, 13, 106
- outliers, 156
- output
 - classic text, 75, 257
 - specifying filename, 283
- OUTPUT command, 384
- output files
 - default directory, 28, 133
- Output window, 176, 231, 234, 283, 288
- overfit, 165

P

- page layout, 60, 243
- page layout preview, 141
- page setup, 141
- Page Setup dialog, 244
- pair-wise correlations, 224

- parent node, 253
 - paste, 290
 - path indicators, 175
 - path references, 29
 - Pearson correlations, 218
 - penalty, 121, 147
 - high-level categorical, 124
 - missing values, 123, 216
 - variable specific, 122
 - PENALTY command, 125, 386
 - Penalty tab, 85, 147, 148, 233
 - predicted probabilities, 78, 175, 260
 - predicted response, 175
 - predicting, 170, 180
 - prediction success table, 61, 67, 245, 250
 - predictor groups, 277
 - predictor variables, 45, 85, 87, 234
 - categorical, 47, 234
 - categorical vs. continuous, 89
 - preparing data, 28
 - primary split, 65, 248
 - primary splitters, 275
 - PRINT command, 389
 - printing
 - gains chart, 143
 - main tree, 60, 243
 - page layout preview, 141
 - page setup, 141, 244
 - preview window, 141
 - reports, 290
 - text output, 283, 284
 - tree, 141
 - tree rules, 183
 - prior probabilities, 219
 - priors, 219
 - DATA, 119
 - EQUAL, 119
 - LEARN, 119
 - MIX, 119
 - SPECIFY, 119
 - specifying, 118
 - TEST, 119
 - PRIORS command, 120, 219, 388
 - Priors tab, 85, 233
 - probability trees, 19, 105
 - Profit tab
 - [Ave. Profit] button, 152
 - [Cum. Ave. Profit] button, 153
 - [Cum. Profit] button, 153
 - [Profit] button, 152
 - Average Profit Learn, 152
 - Default Sort Order, 152
 - Profit Learn, 152
 - Profit Variable, 152
 - programming language, 101
 - progress report, 47, 234
 - prune, 56, 236
 - pruning, 11, 186
 - pruning test method, 165
 - pruning tree, 236
- ## Q
- quartile range, 156
 - QUIT command, 390
- ## R
- random number, 132, 208, 210
 - random sub-sampling, 102
 - rank instability, 187
 - reading ASCII files, 33, 34
 - reading data, 31
 - reading Excel files, 36
 - regression trees, 146, 148
 - relational operators, 409
 - relative contribution, 246
 - relative cost curve, 74, 237
 - relative error, 149
 - REM command, 391
 - repeated cases, 167
 - Report Contents window, 75, 257
 - Report Current menu, 289
 - report details
 - committee of experts, 166
 - Report menu, 42, 229
 - Report All, 289
 - Report Current, 289
 - Set Report Options, 289
 - reporting
 - controlling contents, 129
 - cross-validation results, 131
 - number of competitors, 131
 - number of surrogates, 130
 - options, 289
 - short command notation, 127
 - text reports, 125
 - tree sequence, 131
 - reports
 - box plots, 156
 - classic text output, 75, 257
 - competitors and surrogates, 68, 155, 251
 - node detail, 68, 73, 154, 251, 256
 - node frequency distributions, 71, 253
 - node statistics, 158
 - pre-configured, 289
 - Report Options dialog, 288
 - Report Writer, 288
 - rules, 72, 156, 255
 - splitters, 73, 157, 254
 - target class, 289
 - terminal node detail, 158
 - tree summary, 61, 151, 244
 - viewing rules, 158
 - resampling, 163
 - response statistics tab

- classification, 176
 - regression, 179
- rich text format (.rtf), 290
- robust trees, 186
- ROC
 - graph labels, 127
- ROC curves, 18, 213
- root node splitter
 - specify, 267
- root splits, 154
- rules, 72, 156, 158, 255
 - [All] button, 159
 - [Tagged] button, 159
 - classic, 159
 - SQL, 159
 - viewing, 158
- Rules tab, 156
- running CART, 26
 - permissions, 26

S

- sample data
 - SPAMBASE.CSV, 186
- sample size, 223, 224
 - learn, 113
 - sub-sample, 114
 - test, 113
- Save As ..., 284
- SAVE command, 180, 395
- saving
 - command log, 78, 261
 - committee of experts, 166
 - grove, 257
 - grove file, 171
 - model specifications, 140
 - navigators, 74, 140, 171, 257
 - output, 76, 258
 - reports, 290
 - text output, 283
 - tree topology, 74, 140, 257
- SCORE command, 170, 180, 393
- scoring
 - classification, 176
 - command line, 180
 - data, 77, 259
 - Gains tab, 177
 - GUI output, 176, 178
 - ID variables, 174
 - output data, 175
 - Prediction Success tab, 177
 - proxy target variable, 174
 - regression, 178
 - Response Statistics tab, 176, 179
 - saving predictions, 175
 - saving result to a file, 173
 - Score Data dialog, 173
 - selecting data file, 173

- selecting grove file, 173
 - sub-trees, 174
 - target variable, 174
 - tree sequence, 174
 - weight variable, 174
- scoring models, 170, 172
 - using grove file, 172
 - using navigator file, 172
- SEED command, 132, 396
- select cases, 100
- Select Cases tab, 146, 232
- Select Columns, 196
- Select Columns to Display
 - Direction, 190
 - Fuzzy Match, 190
 - Hide Agreed, 190
 - Rank, 190
- SELECT command, 101, 170, 397
- Select Default Directory dialog, 132
- selecting a tree, 170, 180
- selecting cases, 146
- selecting tree, 236, 244
- selecting variables
 - auxiliary, 85, 91
 - categorical, 85, 88
 - predictors, 85, 87
 - target, 85
- selection criteria, 100
- self-testing, 12
- separation test variable, 99
- setting class names, 92
- setting focus class, 91
- setting up model, 45, 232
- Show Next Pruning, 236
- sorting variable list, 92
- specify root node splitter, 267
- specifying
 - tree type, 86
- specifying model
 - classification, 83
 - regression, 146
- split criteria, 238
- split form
 - categorical, 89
 - continuous, 89
- split value
 - root node, 273
 - setting, 273
- splitter improvement, 69, 252
- Splitter tab, 157
- splitters, 73, 157, 254
 - viewing, 238
- splitting criteria, 13
- splitting rules, 76, 104, 146, 183, 220, 258
 - Class Probability, 104, 105
 - Entropy, 104, 105
 - even splits, 106
 - Gini, 104, 105

- Least Absolute Deviation, 147
 - Least Squares, 147, 149
 - linear combinations, 108
 - Ordered Twoing, 104, 106
 - Symmetric Gini, 104, 105
 - Twoing, 104, 106
 - splitting variable name, 238
 - SQL rules, 159
 - standard error rule, 103
 - starting CART, 26
 - step-wise regression, 222
 - STOP command, 423
 - structured trees, 275
 - SUBMIT command, 399
 - submit command file, 301
 - submit window, 300
 - submitting batch files, 298
 - subsampling, 223
 - size, 114
 - sub-sampling, 287
 - subset of cases, 100
 - sub-trees, 58, 242
 - summary reports, 18, 61, 151, 244
 - Gains Chart tab, 61, 245
 - Misclassification tab, 66, 249
 - node detail, 68, 154, 251
 - Prediction Success tab, 67, 250
 - Profit tab, 152
 - Root Splits tab, 154
 - terminal node detail, 73, 158, 256
 - Terminal Nodes tab, 63, 153, 247
 - Variable Importance tab, 64, 154, 248
 - viewing rules, 158
 - Summary Reports dialog, 61, 244
 - summary statistics, 136
 - surrogate splits, 216
 - surrogate splitters, 275
 - surrogates, 12, 68, 155, 248, 251
 - discount, 248
 - discount weights, 103
 - number of, 103
 - number to report, 130
 - symgini, 13
 - symmetric gini, 105
 - symmetrical cost matrix, 117
 - system requirements
 - minimum, 24
 - recommended, 24
- T**
- tables
 - misclassification, 66, 249
 - prediction success, 67, 250
 - target class, 289
 - target variable, 45, 85, 234
 - class names, 59, 242
 - temporary files
 - default location, 29, 133
 - terminal node distributions, 61, 245
 - Terminal Node Report window, 73, 158, 256
 - terminal node size, 216, 218
 - terminal nodes, 63, 153, 235, 247
 - color coding, 59, 136, 151, 235, 242
 - minimum size, 111
 - test methods, 95, 146
 - fraction of cases, 98
 - no independent testing, 96
 - separation test variable, 99
 - test sample file, 99
 - v-fold cross validation, 96
 - test sample, 90, 99, 211
 - test sample size, 113, 287
 - Testing tab, 146, 147, 232, 233
 - text files, 33
 - text output, 75, 231, 257, 283
 - toolbar icon
 - Command Log, 79, 261
 - Model Setup, 45, 78, 261
 - Options, 125
 - View Data, 290
 - toolbar icons, 42
 - train data, 221
 - Train-Test Consistency, 20, 186
 - transforming variables, 406
 - TRANSLATE command, 182, 400
 - translating, 181
 - translating models, 20, 160, 170, 180
 - choosing output language, 182
 - classic output options, 182
 - command line, 182
 - SAS® options, 182
 - saving result to a file, 182
 - sub-trees, 182
 - tree sequence, 182
 - using grove file, 181
 - using navigator file, 180
 - tree control, 15
 - tree map, 240
 - Tree menu, 42, 229, 236
 - Select Tree, 244
 - Tree Summary Reports..., 61, 244
 - Tree Summary Reports..., 151
 - tree navigator, 149
 - tree sequence, 174, 182, 186, 236
 - number of trees, 131
 - tree size, 236
 - maximum depth, 113, 287
 - maximum number of nodes, 113, 287
 - tree stability, 186
 - Tree Summary Reports, 61, 147, 151, 244
 - tree topology, 48, 149, 235, 236, 244
 - tree type, 86, 88, 234
 - unsupervised, 266
 - Tree window, 240
 - trees

- committee, 167
- ensembles, 162
- initial, 167
- minimum cost, 102
- optimal, 102
- printing, 60, 141, 243
- sub-tree, 58, 242
- viewing, 56, 57, 239, 241

TTC, 20, 186

tutorial, 40

- segmentation, 228

twoing, 13, 106

U

UNIX platform, 296

UNIX usage notes, 310

unsupervised learning, 19, 264

USE command, 402

V

validation

- auto, 19

Var. Imp. tab, 204

- [Box Plot] button, 205
- [Grid] button, 205
- [Max] button, 205
- [Mean] button, 205
- [Median] button, 205
- [Min] button, 205
- [Quartile 0.25] button, 205
- [Quartile 0.75] button, 205
- sort order, 205

variable importance, 16, 64, 103, 154, 248

- contribution of surrogates, 249
- discounting improvement, 248
- measures, 61, 245
- number of surrogates considered, 249

variable names, 32, 33, 35

- ASCII text, 35, 36
- DBMS/COPY, 35

variable transformation, 101

variables

- auxiliary, 85, 91, 134
- categorical, 47, 88, 92, 234
- character, 33
- class names, 59, 242
- contribution, 248
- high-level categorical, 94
- ID, 174
- importance, 248
- number of, 32
- penalize high-level categorical, 121, 124
- penalize improvement, 121
- penalize missing values, 121, 123
- predictors, 45, 85, 87, 234
- selecting, 85, 87, 88

- sorting list, 92
- target, 45, 85, 174, 234
- transforming, 406
- weight, 174

variable-specific penalty, 122

View menu, 42, 59, 229, 236, 238, 242

- Assign Class Names..., 59, 242
- Data Info..., 291
- Node Detail..., 57, 139, 151, 241, 242
- Node Display, 238
- Open Command Log..., 140, 299
- Open Command Log..., 299
- rules, 159
- Rules..., 183, 259
- Show Next Pruning, 236
- Update Command Log, 299
- View Data, 290

viewing

- auxiliary variables information, 134
- data, 290
- data information, 291
- main splitters, 238
- main tree, 57, 241
- sub-tree, 58, 242
- tree, 56, 239
- variable splits, 237

viewing rules, 158

W

warnings and errors, 319

WEIGHT command, 403

weights, 90

- missing, 90
- negative values, 90
- surrogate discount, 103
- zeroed, 90

Window menu, 42, 68, 229, 251

windows

- CART Output, 75, 176, 257
- Data Viewer, 290
- DataInfo Setup, 291
- Main Tree, 56, 57, 239, 241
- Navigator, 48, 149, 235
- Node Report, 68, 251
- Notepad, 300
- Output, 283
- Report Contents, 75, 257
- Splitters, 238
- Sub-Tree, 58, 242
- Terminal Node Report, 73, 158, 256
- Tree Map, 240

Windows keyboard conventions, 43

working directories, 28

workspace usage, 285

X

XYPLOT command, 404

Z

zero case weight, 90

Zoom, 153

Zoom in, 57, 241

Zoom out, 57, 241

z-threshold, 191

z-value, 190